Privacy-Enhancing Optical Embeddings for Lensless Classification

Eric Bezzam[†], Martin Vetterli[†], Matthieu Simeoni[‡]
Audiovisual Communications Laboratory[†], Center for Imaging[‡]
École Polytechnique Fédérale de Lausanne (EPFL)
first.last@epfl.ch

Abstract

Lensless imaging can provide visual privacy due to the highly multiplexed characteristic of its measurements. However, this alone is a weak form of security, as various adversarial attacks can be designed to invert the one-to-many scene mapping of such cameras. In this work, we enhance the privacy provided by lensless imaging by (1) downsampling at the sensor and (2) using a programmable mask with variable patterns as our optical encoder. We build a prototype from a low-cost LCD and Raspberry Pi components, for a total cost of around 100 USD. This very low price point allows our system to be deployed and leveraged in a broad range of applications. In our experiments, we first demonstrate the viability and reconfigurability of our system by applying it to various classification tasks: MNIST, CelebA (face attributes), and CIFAR10. By jointly optimizing the mask pattern and a digital classifier in an end-to-end fashion, low-dimensional, privacy-enhancing embeddings are learned directly at the sensor. Secondly, we show how the proposed system, through variable mask patterns, can thwart adversaries that attempt to invert the system (1) via plaintext attacks or (2) in the event of camera parameters leaks. We demonstrate the defense of our system to both risks, with $55\,\%$ *and* $26\,\%$ *drops in image quality metrics for attacks based on model-based convex optimization and generative neural networks respectively. We open-source a wave propagation and camera simulator needed for end-to-end optimization, the training software, and a library for interfacing with the camera.*

1. Introduction

Cameras are ubiquitous, and they are increasingly connected to cloud services for streaming, post-processing, or inference purposes. The connectivity of such cameras leaves them prone to hacks or data leaks. While strong encryption is important in the digital sphere, we investigate the use of optical and analog hardware as a first layer of privacy. A simple trick to "encrypt" an image at this initial layer is to modify the conventional measurement process, which normally attempts to capture an image as close as possible to the physical scene. To achieve visual privacy, previous work has corrupted the image formation process through: low-resolution measurements [10, 34], replacing the lens with a multiplexing mask [30, 42, 45], and modifying the quantization function [42]. By accounting for this degradation in the optimization of the downstream task (end-to-end training), robust performance can be obtained for a variety of tasks, *e.g.* human activity, pose estimation, face classification [14,34,37,42]. While this approach can provide protection in the case of leaks in measurement data, protection in the event of camera parameter leaks has (to the best of our knowledge) not been investigated. For example, a leak in the camera's point spread function (PSF) makes it possible to recover an image of the underlying object using standard convex optimization techniques from computational imaging [7].

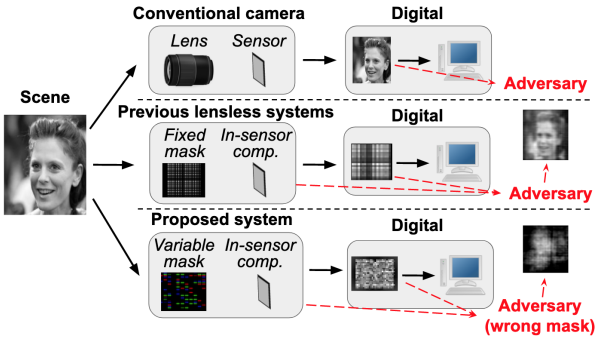


Figure 1. In a conventional camera (top), the optics and hardware try to measure an image as close as possible to the original scene. Leaks in captured data can reveal sensitive information such as identity. To achieve visual privacy, previous work has introduced distortions at the optics and sensor (middle). However, in the event of camera parameters leaks, an adversary can still recover an image of the scene to obtain sensitive information. To protect against both data and system leaks, we propose using a programmable mask with a varying pattern, which makes it difficult for an adversary to invert the degradation performed by the system.

In this work, we propose using a programmable mask with *varying patterns* to provide protection to both data and system parameter leaks. Using a programmable mask allows the system to adapt its encoding function, making it harder for an adversary to decode the normally visually-private measurement in the case of camera parameter leaks. Furthermore, a programmable mask allows for the system to be reconfigured to another inference task. Fig. 1 shows how our proposed system differs from a conventional lensed camera and previous lensless systems that use a fixed modulating mask in place of a lens. Additionally, we propose a physical embodiment of our system that can be put together from cheap and accessible components, totaling at around 100 USD. As a programmable mask, we use a low-cost liquid crystal display (LCD), which costs around 20 USD. We are the first to employ such a device in end-to-end optimization for computational optics. A differentiable digital twin with the selected LCD component is used to optimize the proposed system in simulation. To show the robustness and versatility of our proposed system, we evaluate its performance on several classification tasks: handwritten digit classification (MNIST [23]), gender and smiling binary classification (CelebA [25]), and object classification (CIFAR10 [22]). For all tasks, we find that learning the mask function (as opposed to using a heuristically designed or random encoding) is most resilient to decreases in the embedding dimension, which helps to enhance privacy. To encourage reproducibility, we open source a wave propagation and camera simulator needed for end-to-end optimization,[1] the training software,[2] and library for interfacing with the baseline and proposed cameras.[3]

2. Related work

The work in this paper is at the intersection of multiple disciplines: privacy-preserving imaging, optical computing, and compressive imaging. In this section, we describe how our work differs from the relevant state-of-the-art in the respective fields. Section 3 combines these elements to express our objective as a supervised learning task.

Privacy though optical and analog operations. Privacy-enhancement through degradations in the optical/analog domain has been implemented in previous works by: downsampling at the sensor [10, 34, 42], using a coded-aperture mask [30, 45], and learning a multi-kernel mask [37, 42]. In the latter case, end-to-end optimization is used to jointly learn the mask function and the digital processing through supervised learning, and the learned mask is either printed via lithography techniques or displayed on a spatial light modulator (SLM). To make the trained system robust to adversarial attacks, a privacy term can be added to the loss, namely the performance of adversarial network is minimized while maximizing the performance of the desired task [36,42]. However, the above works assume the privacy-enhancing operations *and adversarial attacks* are fixed. In [18], the authors show that through known-plaintext attacks (obtaining encrypted versions of known inputs), the encoding mask(s) can be obtained to decrypt the visually-private image, raising in question the privacy of such systems. To achieve privacy-enhancing optical embeddings, we combine (1) optical and analog operations (learning a mask pattern and downsampling at the sensor as [42]) with (2) a programmable mask to protect against plaintext attacks and camera parameters leaks.

Optical computing. The use of optical components for data processing – *optical computing* – is a rapidly evolving field, with most works attempting to replace digital operations with optical equivalents, *e.g*. dot products [44], detection/correlation [26], random projections [35], and convolutional neural networks [8]. While having the equivalent digital operation can be convenient, these approaches typically involve bulky, specialized setups that are not compatible or do not scale for mobile and edge device applications, *e.g*. multiple lenses or expensive components such as SLMs (that cost a few thousand USD). Previous work has successfully shown the use of a low cost, LCD to perform optical operations: as a controllable aperture [52] and for single-pixel imaging [16]. However, no work has attempted to use such low-cost components in an end-to-end fashion, nor in a compact design in the spirit of lensless cameras [7]. In this work, the imaging system consists solely of an LCD placed at a short distance from the sensor, and end-to-end optimization is used to determine its pattern.

Compressive imaging and classification. If the sensor resolution is reduced to the extreme, we arrive at a setup akin to single-pixel imaging [12]. While single-pixel imaging requires multiple measurements illuminated with different patterns, mask-based imaging typically needs a single capture. Nonetheless, both share a notion of *sufficient statistics* [11, 12] with regards to tasks such as detection/classification that do not require reconstruction. When reducing the sensor dimension of our proposed system to enhance privacy, we are also interested in the sufficient statistics that are necessary to provide robust classification by learning the best multiplexing from scene to sensor.

3. Deep optics for learning encoding

End-to-end approaches for optimizing optical components, also known as *deep optics* [46], is a recent trend en-

[1] github.com/ebezzam/waveprop; pip install waveprop
[2] github.com/ebezzam/LenslessClassification
[3] github.com/LCAV/LenslessPiCam; pip install lensless

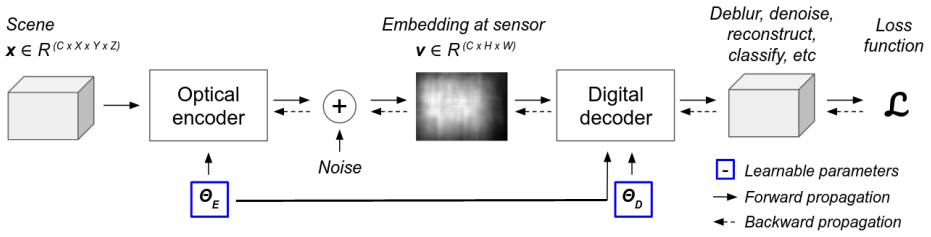


Figure 2. Encoder-decoder perspective of cameras for end-to-end optimization. The scene could be four-dimensional (channel, height, width, depth) whereas the embedding measured at the sensor is at most three-dimensional (channel, height, width).

abled by improved fabrication techniques and the continual development of more powerful and efficient hardware and libraries for machine learning. It is motivated by faster and cheaper inference for edge computing and a desire to co-design the optics and the computational algorithm to obtain optimal performance for a particular application.

An encoder-decoder perspective is often used to frame such end-to-end approaches, casting the optics as the encoder and the subsequent computational algorithm as the decoder, as shown in Fig. 2, and can be formulated as the following optimization problem minimized for a labeled dataset $\{\boldsymbol{x}_i, \boldsymbol{y}_i\}_{i=1}^N$:

$$\hat{\boldsymbol{\theta}}_E, \hat{\boldsymbol{\theta}}_D = \underset{\boldsymbol{\theta}_E, \boldsymbol{\theta}_D}{\operatorname{argmin}} \sum_{i=1}^{N} \mathcal{L}\Big(\boldsymbol{y}_i, \underbrace{D_{\boldsymbol{\theta}_E, \boldsymbol{\theta}_D}\big(\overbrace{O_{\boldsymbol{\theta}_E}(\boldsymbol{x}_i)}^{\text{embedding } \boldsymbol{v}_i}\big)}_{\text{decoder output } \hat{\boldsymbol{y}}_i}\Big). \quad (1)$$

$\boldsymbol{\theta}_E, \boldsymbol{\theta}_D$ are the optical encoder and digital decoder parameters that we seek to optimize for a given task. The optical encoder $O_{\boldsymbol{\theta}_E}(\cdot)$ encapsulates propagation in free space and through all optical components prior to the sensor, and downsampling and additive noise at the sensor. Given a scene $\boldsymbol{x}_i$, it outputs the sensor embedding $\boldsymbol{v}_i$. $D_{\boldsymbol{\theta}_E, \boldsymbol{\theta}_D}(\cdot)$ is the digital decoder (*e.g.* a neural network), which can perform a whole slew of tasks: deblurring, denoising, image reconstruction, classification, *etc*. It can optionally make use of the optical encoder parameters, *e.g.* the PSF can be useful for image deconvolution/reconstruction. The decoder's output is fed to an application-dependent loss function $\mathcal{L}(\cdot)$ along with the ground-truth output $\boldsymbol{y}_i$, and the error is back-propagated to update $\boldsymbol{\theta}_E, \boldsymbol{\theta}_D$.

Within the context of privacy-preserving tasks, a common approach is to apply adversarial training [15, 33, 36, 42, 48], namely simultaneously minimize a task-related loss such as Eq. (1) and maximize a privacy-related term, *e.g.* the classification or reconstruction error of an adversarial network. Similar to [10, 34, 45], we choose not to incorporate an adversarial component in our training, since the latter would only enhance privacy for the *fixed adversarial attacks* it assumes, resulting in potential blind spots – in particular for large datasets with complicated structures [50]. Moreover, as we shall soon demonstrate, the design of our camera already produces such degenerate measurements that an attacker with knowledge about the camera system would struggle to reconstruct an image of the original scene. This is achieved by means of a (re)programmable mask component that randomly changes its configuration to mitigate potential privacy-infringing attacks on the system. Such a design is motivated by the findings of Jiao *et al*. [18], who show that decrypted images with the wrong mask appear noise-like and that no information about the actual image can be visually perceived or recovered. This is confirmed by Fig. 5, which shows reconstructions when a *correct* and an *incorrect* mask are used for decoding simulated images with our system (top and bottom row respectively).

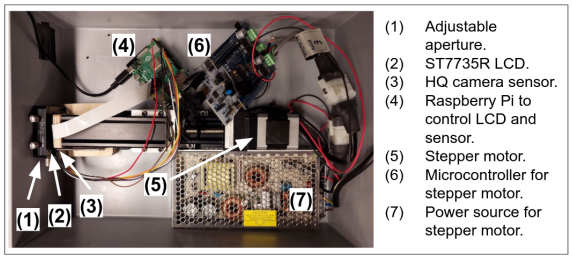


Figure 3. Experimental prototype of programmable, mask-based lensless camera.

4. Proposed system

4.1. Hardware design

Our camera design is motivated by the benefits of lensless cameras (compact, low-cost, visual privacy) and programmability. To this end, a programmable mask serves as the only optical component in our encoder, specifically an off-the-shelf LCD driven by the ST7735R device which can be purchased for $20.[4] An experimental prototype of the proposed design can be seen in Fig. 3. The LCD is wired to a Raspberry Pi ($35) with the Raspberry Pi High Quality 12.3 MP Camera ($50) as a sensor, totaling our design to just $105. The prototype includes an adjustable aperture, and a stepper motor for programmatically setting the dis-

[4] https://www.adafruit.com/product/358

tance between the LCD and the sensor, both of which can be removed to produce a more compact design.

4.2. Digital twin for training

End-to-end optimization requires a sufficiently accurate and differentiable simulation of the physical setup. Our digital twin of the imaging system accounts for wave-based image formation for spatially incoherent, polychromatic illumination, as is typical of natural scenes. A simulation based on wave optics (as opposed to ray optics) is necessary to account for diffraction due to the small apertures of the mask and for wavelength-dependent propagation. To determine whether a wave-optics simulation is necessary, the Fresnel number N_F can be used, with ray optics generally requiring $N_F \gg 1$ [7] . For our setup, $N_F \in [1.2, 2]$ suggesting that diffraction effects need to be explicitly accounted for.[5]

In our setup, the scene of interest is at a fixed distance d_1 from the optical encoder, which itself is at a distance d_2 from the image plane. We adopt a common assumption from *scalar diffraction theory*, namely that image formation is a linear shift-invariant (LSI) system between two parallel planes for a given wavelength λ [13]. Consequently, there exists an impulse response that can be convolved with the *scaled* scene to obtain its image at a given distance z:

$$I_2(\boldsymbol{x}; z, \lambda) = \int_{\mathbb{R}^2} d\boldsymbol{r}\, p(\boldsymbol{x} - \boldsymbol{r}; z, \lambda) \left[\frac{1}{|M|^2} I_0\Big(\frac{\boldsymbol{r}}{M}; \lambda\Big) \right], \tag{2}$$

where I_0 and I_2 are the intensities at the scene and image planes respectively, $\boldsymbol{x} \in \mathbb{R}^2$ and $\boldsymbol{r} \in \mathbb{R}^2$ are coordinates on the image and scene planes respectively, $M = -d_2/d_1$ is a magnification/inversion factor, and p is the intensity PSF [13]. As we have an incoherent illumination, the imaging is linear in intensity, lending to the above convolutional relationship. Our digital twin modeling amounts to obtaining a PSF that encapsulates propagation from a given plane in the scene to the sensor plane. There are two ways to obtain this PSF: measuring it with a physical setup or simulating it. Below we describe how it can be simulated for a mask-based encoder as in our proposed system.

4.3. Simulating the PSF for a mask-based encoder

We model a programmable mask as a superposition of apertures for each adjustable pixel:

$$M(\boldsymbol{x}) = \sum_{k}^{K} w_k A(\boldsymbol{x} - \boldsymbol{x}_k), \tag{3}$$

[5]The Fresnel number is given by $N_F = a^2/d\lambda$, where a is the size of the mask's open apertures, d the propagation distance, and λ the wavelength. For our setup, $a = 0.06\,\text{mm}$, $d = 4\,\text{mm}$, and $\lambda \in [450\,\text{nm}, 750\,\text{nm}]$.

where the complex-valued weights $\{w_k\}_{k=1}^K$ satisfy $|w_k| \leq 1$, the coordinates $\{(\boldsymbol{x}_k)\}_{k=1}^K$ are the centers of the mask pixels, and the aperture function $A(\boldsymbol{x})$ is assumed to be identical for each pixel. The weights $\{w_k\}_{k=1}^K$ correspond to the mask pattern we would like to determine for a given task. However, the intensity PSF of the mask function, and not the mask function itself, is needed to obtain the embedding of a scene as given by Eq. (2). The intensity PSF is the squared magnitude of the amplitude PSF [13], namely $p(\boldsymbol{x}; z, \lambda) = |h(\boldsymbol{x}; z, \lambda)|^2$. The amplitude PSF for our setup can be modeled as convolution between (1) spherical waves impinging on the mask and (2) the free-space propagation kernel. Computationally, it is best performed in the spatial frequency domain (via FFTs) as

$$h(\boldsymbol{x}; z = d_1 + d_2, \lambda) = \mathcal{F}^{-1}\Big(\mathcal{F}\Big(M(\boldsymbol{x}) e^{j\frac{2\pi}{\lambda}\sqrt{\|\boldsymbol{x}\|_2^2 + d_1^2}}\Big) \times H(\boldsymbol{u}; z = d_2, \lambda)\Big), \tag{4}$$

where $\mathcal{F}$ and $\mathcal{F}^{-1}$ denote the spatial Fourier transform and its inverse, $H(\boldsymbol{u}; z, \lambda)$ is the free-space propagation frequency response [28] (defined in Eq. (A.9)), and $\boldsymbol{u} \in \mathbb{R}^2$ are spatial frequencies of $\boldsymbol{x}$. In Appendix A, we describe in more detail the modeling of the PSF, programmable masks, and specifics for the ST7735R component.

5. Experiments

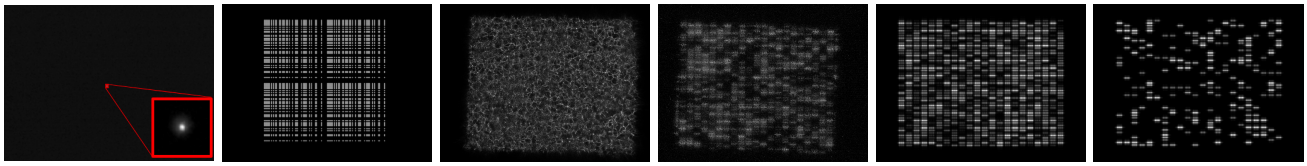
In this section, we apply our proposed camera and end-to-end optimization to various classification tasks. We optimize a classifier that directly operates on the camera measurement, rather than introducing an (unnecessary) intermediate reconstruction step as in [43]. Consequently, the digital decoder $D_{\boldsymbol{\theta}_D}(\cdot)$ in Eq. (1) does not require explicit information from the encoder.

For our proposed system with the ST7735R component, the learnable optical encoder parameters $\boldsymbol{\theta}_E$ for Eq. (1) are the mask weights $\{w_k\}_{k=1}^K$ in Eq. (3), which are restricted to be real and non-negative for amplitude modulation.

We conduct the following experiments to demonstrate:

1. Sections 5.1 to 5.3: the viability of the proposed, low-cost system and end-to-end optimization of (1) the mask weights and (2) the digital classifier on multiple tasks: handwritten digits (MNIST [23]), gender and smiling (CelebA [25]), and objects (CIFAR10 [22]) classification.
2. Section 5.4: the proposed system's defense (low-dimensional measurements and variable mask patterns) against parameters leaks and plaintext attacks that try to recover the underlying image.

In our experiments, we consider various architectures for the digital decoder: logistic regression (LR), a fully-connected neural network with a single hidden layer of 800



(a) Lens. (b) Coded aperture. (c) Diffuser. (d) Fixed mask (m). (e) Fixed mask (s). (f) Learned mask.

Figure 4. Point spread functions (PSFs) for baseline and proposed cameras. *Learned mask* is unique for each embedding dimension, digital decoder, and task. The one shown in Fig. 4f was optimized for an embedding dimension of (24×32), a two-layer fully connected neural network, and for gender classification. See Appendix J for the learned PSFs for other sensor dimensions, digital decoders, and tasks.

units (FC), and VGG11 [40]. Further details on model architectures, as well as training hardware and software, can be found in Appendix D. Training hyperparameters, example images, and train-test accuracy curves for each experiment can be found in Appendices E to H.

Datasets. For MNIST, we use the provided train-test split: $60'000$ training and $10'000$ test examples. Referring to Eq. (1), the $\{\boldsymbol{x}_i\}_{i=1}^N$ coming from MNIST are (28×28) images of handwritten digits and $\{\boldsymbol{y}_i\}_{i=1}^N$ one-hot encoded vectors corresponding to the labels from 0 to 9.

For CelebA, we use a subset of $100'000$ examples from the original dataset of $202'599$ images for training the optical encoder, allocating 85% for training and 15% for testing. This split results in a gender distribution of $58\,\%$ female and $42\,\%$ male, and $52\,\%$ not smiling and $48\,\%$ smiling in both the train and test set. Gender classification is chosen as it represents a global feature while smiling classification is localized. As these tasks are binary, the outputs are scalar, namely $\{y_i\}_{i=1}^N \in \{0, 1\}$. The original images are (178×218) and RGB, but we convert them to grayscale. For the privacy-preserving experiments in Section 5.4, we use a separate subset of $100'000$ examples.

For CIFAR10, we use the provided train-test split: $50'000$ training and $10'000$ test examples. The original images are (32×32) and RGB. As there are 10 classes, $\{\boldsymbol{y}_i\}_{i=1}^N$ are length-10 one-hot encoded vectors.

Baseline and proposed cameras. To produce the embeddings $\{\boldsymbol{v}_i\}_{i=1}^N$ in Eq. (1), the original images $\{\boldsymbol{x}_i\}_{i=1}^N$ are simulated according to Alg. 1. We compare six imaging systems: *Lens*, *Coded aperture* [2], *Diffuser* [1], *Fixed mask (m)*, *Fixed mask (s)*, and *Learned mask*. *Lens* shows how a non-visually-private system would perform, while *Coded aperture* and *Diffuser* represent typical mask design strategies: a binary amplitude modulation scheme that simplifies reconstruction and a phase modulation mask respectively. *Fixed mask (m)* and *Fixed mask (s)* correspond to the proposed imaging system discussed in Section 4, but with a randomly-set mask pattern. *Fixed mask (m)* is measured with hardware, while *Fixed mask (s)* is a simulated to gauge how our digital twin compares with a measured PSF. *Learned mask* corresponds to the proposed imaging system with learning the mask weights in Eq. (3), *i.e.* $\boldsymbol{\theta}_E = \{w_k\}_{k=1}^K$ for Eq. (1). In contrast, all other investigated imaging systems are *fixed optical encoders*, with *frozen* parameters $\boldsymbol{\theta}_E$.

Algorithm 1 Simulating the optical encoder $O_{\boldsymbol{\theta}_E}(\cdot)$ in Eq. (1). See Appendix B for more detailed description.

Require: Original image $\boldsymbol{x} \in \mathbb{R}^{C \times H \times W}$, PSF $\boldsymbol{p} \in \mathbb{R}^{C \times H_{\text{PSF}} \times W_{\text{PSF}}}$, object height h_{obj}, scene-to-encoder distance d_1, encoder-to-sensor distance d_2, downsampling factor $D \geq 1$, SNR σ

Ensure: At sensor $\boldsymbol{v} \in \mathbb{R}^{C \times DH_{\text{PSF}} \times DW_{\text{PSF}}}$

$\boldsymbol{x}_{\text{scene}} \leftarrow \text{Prep}(\boldsymbol{x}, h_{\text{obj}}, d_1, d_2, \boldsymbol{p})$ ▷ *Pad image according to physical setup and sensor aspect ratio, and resize to PSF dimensions.*

$\boldsymbol{v} \leftarrow \boldsymbol{x}_{\text{scene}} * \boldsymbol{p}$ ▷ *Convolve each channel to obtain measurement at sensor.*

if $D > 1$ **then**

  $\boldsymbol{v} \leftarrow \text{Downsample}(\boldsymbol{v}, D)$

end if

$\boldsymbol{v} \leftarrow \text{ShotNoise}(\boldsymbol{v}, \sigma)$

For each camera, a PSF is needed to perform the simulation procedure described in Alg. 1. The PSFs for *Lens*, *Diffuser*, and *Fixed mask (m)* are measured with hardware, and the rest are simulated. The learnable parameters of *Learned mask* are updated at each batch to obtain a new simulated PSF, as described in Section 4.3. Further technical details, such as the components for the measured PSFs and parameters for the simulated PSFs, can be found in Appendix C. The PSFs themselves are shown in Fig. 4.

5.1. MNIST proof-of-concept

Varying embedding dimension. Tab. 1 reports the best test accuracy for each optical encoder, for a varying sensor embedding dimension and for two classification architectures: LR and FC. While all approaches decrease in performance as the dimension reduces, *Learned mask* is the most resilient as quantified by *Relative drop* in Tab. 1. While the performance gap between *Learned mask* and fixed lens-

Table 1. MNIST accuracy on test set, simulated accordingly for each camera and dimension. *Relative drop* indicates relative drop in accuracy for each camera (for a given architecture) from (24×32) to (3×4).

Classifier →	Logistic regression					Two-layer fully connected, 800 hidden units				
Embedding → *Encoder* ↓	24×32 =768	12×16 =192	6×8 =48	3×4 =12	*Relative drop*	24×32 =768	12×16 =192	6×8 =48	3×4 =12	*Relative drop*
Lens	*92.4%*	*75.2%*	*41.7%*	*21.9%*	*76.3%*	*98.4%*	*83.8%*	*40.7%*	*22.5%*	*77.1%*
Coded aperture	82.4%	87.2%	71.5%	57.3%	30.5%	97.7%	97.0%	92.2%	71.2%	27.2%
Diffuser	91.7%	82.3%	76.6%	54.5%	40.6%	98.6%	98.6%	97.0%	78.9%	20.0%
Fixed mask (m)	92.8%	91.9%	87.7%	73.4%	21.0%	**98.7%**	**98.7%**	97.8%	87.5%	11.4%
Fixed mask (s)	92.7%	91.8%	86.5%	72.7%	21.6%	98.6%	**98.7%**	97.5%	88.6%	10.2%
Learned mask	**95.5%**	**93.5%**	**91.4%**	**79.3%**	**17.0%**	**98.7%**	98.4%	**98.0%**	**91.7%**	**7.10%**

Table 2. Relative drop in MNIST accuracy for two-layer fully connected network with 800 hidden units and an embedding dimension of (24×32) for a randomly transformed data. See Tab. F.3 for absolute performance. Lower is better.

	Shift	Rescale	Rotate	Perspective
Lens	*12.1%*	*13.5%*	*3.20%*	*13.5%*
Coded aperture	77.2%	15.3%	6.80%	67.9%
Diffuser	67.8%	2.20%	**3.00%**	27.6%
Fixed mask (m)	56.7%	2.00%	3.20%	18.3%
Fixed mask (s)	61.6%	**1.90%**	3.10%	19.8%
Learned mask	**30.4%**	6.40%	**3.00%**	**14.4%**

less encoders decreases when a digital classifier with more parameters is used (FC), the benefits of learning this multiplexing are clear for a very low embedding dimension of (3×4). Fig. E.6 shows example sensor embeddings for various camera and embedding dimensions pairs.

Robustness to common image transformations. The MNIST dataset is size-normalized and centered, making it ideal for training and testing image classification systems but is not representative of how images may be taken in-the-wild. In this experiment, we evaluate the robustness of lensless encoders to common image transformation: shifting, rescaling, rotating, and perspective changes. The range of values for each transformation, and augmented examples can be found in Appendix F.

Tab. 2 reports the relative drop in classification accuracy for each optical encoder and for each image transformation when using the FC architecture and an embedding dimension of (24×32), namely when all models perform rather equivalently. The main difficulty for lensless cameras is shifting as the whole sensor no longer captures multiplexed information on the entire sensor (see Appendix F.1 for example embeddings). This leads to a significant reduction in classification accuracy for all lensless approaches. *Learned mask* is able to cope with shifting much better than the fixed encoding strategies for lensing imaging, perhaps because it can adapt its multiplexing for such perturbations. For rescaling and rotating, we observe no benefit in performance in learning the mask pattern. Moreover, the multiplexing characteristic of lensless cameras allows them to be more robust to rescaling than a classical lensed camera.

5.2. Face attribute classification (CelebA)

For this experiment, we apply the baseline and proposed cameras to a dataset and task that may benefit from privacy-enhancing features: face classification. Precisely, we target two tasks – gender and smiling binary classification – a global and local feature respectively. The left and middle sections of Tab. 3 report the best test accuracy for both tasks and for two embedding dimensions – (24×32) and (3×4). Out of the lensless approaches, *Learned mask* is best performing. Similar to MNIST, the relative drop in accuracy – from (24×32) to (3×4) – again shows how *Learned mask* is the most robust in performance if one is interested in improving the privacy by decreasing the sensor resolution. Fig. G.13 shows example sensor embeddings as the dimension decreases.

5.3. RGB object classification (CIFAR10)

For this experiment, we address a more complicated task: object classification from RGB data. To this end, we apply a more powerful model at the digital end: VGG11. The rightmost section of Tab. 3 reports the best test accuracy for a varying sensor embedding. Among the lensless approaches, *Learned mask* performs the best. However, the performance gap with the non-private *Lens* is large, which may need to be addressed with a more potent optical encoder or digital classifier. The confusion matrices in Fig. H.17 show which classes lensless encoders struggle to distinguish, most notably *cats-dogs*. Fig. H.18 shows example sensor embeddings as the dimension decreases.

Table 3. Classification results for gender and smiling classification (CelebA [25]) and object classification (CIFAR10 [22]). Gender and smiling classification with CelebA use a two-layer fully-connected neural network with a hidden layer of 800 units as a digital classifier, while object classification with CIFAR10 uses VGG11. *Relative drop* indicates relative drop in accuracy from (24×32) to (3×4).

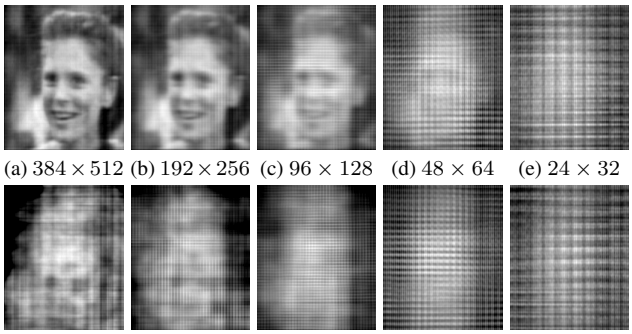
Task →	Gender (FC)			Smiling (FC)			CIFAR10 (VGG11)			
Embedding → *Encoder* ↓	24×32 =768	3×4 =12	*Relative drop*	24×32 =768	3×4 =12	*Relative drop*	3×27×36 =2916	3×13×17 =663	3×6×8 =144	3×3×4 =36
Lens	*91.2%*	*71.1%*	*22.1%*	*89.3%*	*56.9%*	*36.3%*	*86.8%*	*79.8%*	*66.4%*	*46.5%*
Coded aperture	83.9%	61.8%	26.3%	84.2%	56.1%	33.4%	47.5%	51.5%	49.1%	38.6%
Diffuser	87.4%	67.8%	22.4%	86.1%	61.4%	28.7%	45.7%	45.3%	43.0%	39.1%
Fixed mask (m)	89.9%	71.1%	20.9%	87.5%	63.4%	27.6%	51.3%	49.0%	44.7%	41.0%
Fixed mask (s)	90.9%	68.9%	24.2%	87.7%	63.1%	28.0%	50.8%	48.3%	46.8%	38.9%
Learned mask	**91.0%**	**80.7%**	**11.3%**	**88.4%**	**76.0%**	**14.0%**	**63.0%**	**61.6%**	**56.9%**	**48.2%**

5.4. Defense against leaks and plaintext attacks

While the multiplexing property of lensless cameras combined with lowering the embedding dimension appear to enhance the *visual* privacy of the embeddings, it does not imply that a computational imaging technique cannot recover a discernible image. In the following experiments, we assume that an adversary has obtained a set of visually private measurements from our proposed system. We demonstrate and quantify how the joint effect of (1) lowering the sensor resolution and (2) using a programmable mask enhances the privacy of the resulting embeddings.

Leak of optical encoder parameters. Given a lensless camera measurement, a malicious user could formulate an inverse problem in order to recover the underlying scene (see Appendix I.1 for formulation). Such an attack would require an estimate of the PSF which could be obtained two ways: (1) there was a leak in mask values and an estimate PSF could be simulated using the approach described in Section 4.3 or (2) through a plaintext attack that reveals the mask function directly at the sensor. Note that the latter is less realistic since it requires that the adversary control the lighting in which the camera is placed (to emit a single point source) and that they can measure the response at a sufficiently high resolution.

Assuming that the malicious user is able to obtain an accurate estimate of the PSF, we can observe in Fig. 5 reconstructed images as the sensor resolution decreases: from (384× 512) in Fig. 5a down to (24×32) in Fig. 5e. As the resolution decreases, the quality of the recovered scene deteriorates: at (24×32) it is impossible to discern a face. In the last row of Fig. 5, we emulate the scenario that the adversary uses a wrong estimate of the PSF: either the mask values were reconfigured since the leak, or the adversary simply guessed the mask values when simulating the PSF. We can observe the recovered scenes resemble noise (similar to the findings of [18] when trying to perform single-pixel imaging with the wrong illuminations).



(a) 384×512 (b) 192×256 (c) 96×128 (d) 48×64 (e) 24×32

(f) 384×512 (g) 192×256 (h) 96×128 (i) 48×64 (j) 24×32

Figure 5. Discerning content from lensless camera raw measurements is next to impossible, motivating privacy-preserving imaging with such cameras (see example measurements in Fig. G.13). However, with sufficient knowledge about the camera, *e.g.* a point spread function (PSF), and an appropriate computational algorithm, one is able to recover an estimate of the underlying object (top row, see Appendix I.1 for inverse problem formulation). As the raw sensor measurement is increasingly under-sampled, it becomes more and more difficult to recover a meaningful estimate of the underlying object. The caption indicates the resolution of the raw measurement, while all recovered images are at the resolution of the PSF – (384×512). Bottom row demonstrates the recovery estimate when an incorrect PSF is used, motivating our use of a programmable mask to defend against leaks of camera system parameters.

Tab. 4 quantifies the quality of reconstruction for varying embedding dimensions and knowledge of the PSF. When a good estimate of the PSF can be obtained, the multiplexing property of lensless cameras may not be enough to preserve privacy, but can be when coupled with a decrease in sensor resolution. With a bad estimate of the PSF, it is not possible to recover a meaningful image of the underlying scene. Through the use of a programmable mask, the system can protect against leaks in mask values, and even allow imaging at higher resolutions if the task at hand requires it.

Table 4. Image quality metrics (PSNR/SSIM) of reconstruction via convex optimization for varying sensor measurement resolution and when using a bad PSF estimate (see Appendix I.1 for inverse problem formulation). Results are averaged over 50 files. Higher is better. Yellow hightlight indicates a discernible image, see Fig. 5 for examples. 47 % / 55 % drop in metrics for (384×512).

PSF	384×512	192×256	96×128	48×64	24× 32
Good	22.6 / 0.80	19.7 / 0.67	16.0 / 0.52	12.9 / 0.26	11.7 / 0.23
Bad	11.9 / 0.36	11.7 / 0.25	12.1 / 0.28	11.9 / 0.20	11.4 / 0.22

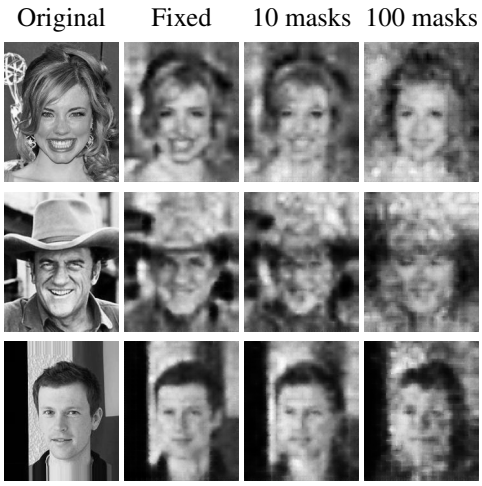


Figure 6. Example outputs of a decoder that was trained with $100'000$ plaintext attacks of embeddings of resolution (24×32). See Appendix I.2 for more example outputs.

Table 5. Image quality (PSNR/SSIM) of trained decoder for varying number of plaintext attacks and number of varying masks. Higher is better. 17 % / 26 % drop in metrics for $100'000$.

# attacks ↓	Fixed mask	10 masks	100 masks
100	13.9 / 0.26	12.5 / 0.21	11.8 / 0.20
$1'000$	16.3 / 0.40	14.2 / 0.32	13.4 / 0.29
$10'000$	18.1 / 0.53	16.2 / 0.43	14.6 / 0.38
$100'000$	19.3 / 0.61	18.0 / 0.53	16.1 / 0.45

Plaintext attacks on compromised camera. If obtaining an estimate of the PSF proves to be too difficult, an adversary could attempt to learn a decoder through a series of plaintext attacks, namely collect the corresponding embeddings for a set of known images. For this scenario, the adversary should (1) know the data that the system is configured to encode, and (2) be able to obtain the sensor embeddings of their set of known images. A defense against a series of such plaintext attacks could be to vary the mask pattern, *e.g.* with a programmable mask as in our proposed system, so that the dataset collected from the adversary is a combination of embeddings from different projections rather than just a single fixed projection. Note that at inference a different classifier would need to be used as the mask and classifier are jointly trained. The hypothesis is that learning a decoder for different projections is more difficult, as the decoder needs to learn the high entropic distribution of multiple masks.

In Tab. 5 we present results that confirm this hypothesis. For a sensor resolution of (24×32) – where the previous attack failed to produce an image where a face could be discerned (see Fig. 5e) – we train a decoder for a varying number of plaintext attacks and number of masks used at capture. We train a convolutional network similar to that of generative adversarial networks [19],[6] see Appendix I.2 for architecture and training details.

In the previous experiment, the identity could be discerned for a PSNR and SSIM of 16.0 and 0.52 respectively (resolution of (96×128) as seen in Fig. 5c). The scenarios with similar (or better) metrics are highlighted in yellow. As the number of varying masks increases (1 to 100), we see that it becomes increasingly difficult for the decoder to produce a meaningful image. More plaintext attacks are needed which becomes impractical for an adversary to collect. Fig. 6 shows examples of reconstructed outputs.

6. Conclusion

We have proposed a low-cost, privacy-enhancing system for performing lensless imaging with an off-the-shelf LCD component. Leveraging an end-to-end optimization procedure to jointly learn the mask pattern and a reconstruction-free classifier, we have demonstrated its ability to outperform previous approaches that use heuristic or random mask patterns. A notable feature of the proposed system is the use of a programmable mask to mitigate data and camera parameter leaks, and therefore enhance privacy. By varying the mask pattern, the system can thwart an adversary that attempts to invert the lensless encoding. As future work, it would be interesting to investigate how the proposed programmable mask strategy thwarts adversarial attacks that seek to identify attributes, *e.g.* ethnicity and age group, rather than reconstructing an image.

Limitations. A single layer in the optical encoder limits its abilities to learn rich embeddings. Increasing the number of layers could improve performance [24], but at the expense of a less compact system and a more involved learning procedure. Moreover, relying on physical devices for computation can have drawbacks. They are more susceptible to degradation (due to usage and over time) than purely digital computations. Finally, device tolerances can lead to unwanted differences between two seemingly identical setups. Such differences may be more prominent for low-cost

[6] We also tried fine-tuning a pre-trained StyleGAN2 [20] but found the generated images were better when training a regressor from scratch.

components such as the cheap LCD used in this paper, as opposed to commercial SLMs.

Funding. This work was in part funded by the Swiss National Science Foundation (SNSF) under grants 200021_181978/1 "SESAM - Sensing and Sampling: Theory and Algorithms" (E. Bezzam) and CRSII5 193826 "AstroSignals - A New Window on the Universe, with the New Generation of Large Radio-Astronomy Facilities" (M. Simeoni).

References

[1] Nick Antipa, Grace Kuo, Reinhard Heckel, Ben Mildenhall, Emrah Bostan, Ren Ng, and Laura Waller. Diffusercam: lensless single-exposure 3d imaging. *Optica*, 5(1):1–9, Jan 2018. 5, 15, 16, 20, 22, 25, 26

[2] M. Salman Asif, Ali Ayremlou, Aswin Sankaranarayanan, Ashok Veeraraghavan, and Richard G. Baraniuk. Flatcam: Thin, lensless cameras using coded aperture and computation. *IEEE Transactions on Computational Imaging*, 3(3):384–397, 2017. 5, 16, 17, 20, 22, 25

[3] Eric Bezzam, Sepand Kashani, Paul Hurley, Martin Vetterli, and Matthieu Simeoni. pyffs: A python library for fast fourier series computation and interpolation with gpu acceleration. *SIAM Journal on Scientific Computing*, 44(4):C346–C366, 2022. 12

[4] Eric Bezzam, Sepand Kashani, Martin Vetterli, and Matthieu Simeoni. LenslessPiCam: A hardware and software platform for lensless computational imaging with a Raspberry Pi, 2022. 17

[5] C. Biscarrat, S. Parthasarathy, G. Kuo, and N. Antipa. Build your own diffusercam: Tutorial, 2018. 17

[6] Vivek Boominathan, Jesse K. Adams, Jacob T. Robinson, and Ashok Veeraraghavan. Phlatcam: Designed phase-mask based thin lensless camera. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 42(7):1618–1629, 2020. 15

[7] Vivek Boominathan, Jacob T Robinson, Laura Waller, and Ashok Veeraraghavan. Recent advances in lensless imaging. *Optica*, 9(1):1–16, 2022. 1, 2, 4

[8] Julie Chang, Vincent Sitzmann, Xiong Dun, Wolfgang Heidrich, and Gordon Wetzstein. Hybrid optical-electronic convolutional neural networks with optimized diffractive optics for image classification. *Scientific reports*, 8(1):1–10, 2018. 2

[9] Patrick L. Combettes and Jean-Christophe Pesquet. *Proximal Splitting Methods in Signal Processing*, pages 185–212. Springer New York, New York, NY, 2011. 26

[10] Ji Dai, Behrouz Saghafi, Jonathan Wu, Janusz Konrad, and Prakash Ishwar. Towards privacy-preserving recognition of human activities. In *2015 IEEE International Conference on Image Processing (ICIP)*, pages 4238–4242, 2015. 1, 2, 3

[11] Mark A. Davenport, Marco F. Duarte, Michael B. Wakin, Jason N. Laska, Dharmpal Takhar, Kevin F. Kelly, and Richard G. Baraniuk. The smashed filter for compressive classification and target recognition. In *Computational Imaging V*, volume 6498, page 64980H. International Society for Optics and Photonics, SPIE, 2007. 2

[12] Marco F. Duarte, Mark A. Davenport, Dharmpal Takhar, Jason N. Laska, Ting Sun, Kevin F. Kelly, and Richard G. Baraniuk. Single-pixel imaging via compressive sampling. *IEEE Signal Processing Magazine*, 25(2):83–91, 2008. 2

[13] J.W. Goodman. Introduction to Fourier optics, 2005. 4, 12, 13

[14] Carlos Hinojosa, Juan Carlos Niebles, and Henry Arguello. Learning privacy-preserving optics for human pose estimation. In *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 2553–2562, 2021. 1

[15] Carlos Hinojosa, Juan Carlos Niebles, and Henry Arguello. Learning privacy-preserving optics for human pose estimation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 2573–2582, 2021. 3

[16] Gang Huang, Hong Jiang, Kim Matthews, and Paul Wilford. Lensless imaging by compressive sensing. In *2013 IEEE International Conference on Image Processing*, pages 2101–2105, 2013. 2

[17] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15, page 448456. JMLR.org, 2015. 17

[18] Shuming Jiao, Ting Lei, Yang Gao, Zhenwei Xie, and Xiaocong Yuan. Known-plaintext attack and ciphertext-only attack for encrypted single-pixel imaging. *IEEE Access*, 7:119557–119565, 2019. 2, 3, 7

[19] Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. Progressive growing of gans for improved quality, stability, and variation. *arXiv preprint arXiv:1710.10196*, 2017. 8, 26

[20] Tero Karras, Miika Aittala, Janne Hellsten, Samuli Laine, Jaakko Lehtinen, and Timo Aila. Training generative adversarial networks with limited data. In *Proc. NeurIPS*, 2020. 8, 26

[21] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. 18

[22] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009. 2, 4, 7

[23] Yann LeCun. The mnist database of handwritten digits. *http://yann. lecun. com/exdb/mnist/*, 1998. 2, 4, 14

[24] Xing Lin, Yair Rivenson, Nezih T Yardimci, Muhammed Veli, Yi Luo, Mona Jarrahi, and Aydogan Ozcan. All-optical machine learning using diffractive deep neural networks. *Science*, 361(6406):1004–1008, 2018. 8

[25] Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. Deep learning face attributes in the wild. In *Proceedings of International Conference on Computer Vision (ICCV)*, December 2015. 2, 4, 7, 26

[26] A.V. Lugt. Signal detection by complex spatial filtering. *IEEE Transactions on Information Theory*, 10(2):139–145, 1964. 2

[27] Eric Markley, Fanglin Linda Liu, Michael Kellman, Nick Antipa, and Laura Waller. Physics-based learned diffuser for single-shot 3d imaging. In *NeurIPS 2021 Workshop on Deep Learning and Inverse Problems*, 2021. 15, 16

[28] Kyoji Matsushima and Tomoyoshi Shimobaba. Band-limited angular spectrum method for numerical simulation of free-space propagation in far and near fields. *Opt. Express*, 17(22):19662–19673, Oct 2009. 4, 12

[29] Richard P. Muffoletto, John M. Tyler, and Joel E. Tohline. Shifted fresnel diffraction for computational holography. *Opt. Express*, 15(9):5631–5640, Apr 2007. 12

[30] Thuong Nguyen Canh and Hajime Nagahara. Deep compressive sensing for visual privacy protection in flatcam imaging. In *2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW)*, pages 3978–3986, 2019. 1, 2

[31] 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. 17

[32] Y. Peng, S. Choi, N. Padmanaban, and G. Wetzstein. Neural Holography with Camera-in-the-loop Training. *ACM Trans. Graph. (SIGGRAPH Asia)*, 2020. 13, 14, 15

[33] Francesco Pittaluga, Sanjeev Koppal, and Ayan Chakrabarti. Learning privacy preserving encodings through adversarial training. In *2019 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 791–799. IEEE, 2019. 3

[34] Michael S. Ryoo, Brandon Rothrock, Charles Fleming, and Hyun Jong Yang. Privacy-preserving human activity recognition from extreme low resolution. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, AAAI'17, page 42554262. AAAI Press, 2017. 1, 2, 3

[35] A. Saade, F. Caltagirone, I. Carron, L. Daudet, A. Drémeau, S. Gigan, and F. Krzakala. Random projections through multiple optical scattering: Approximating kernels at the speed of light. In *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, page 62156219. IEEE Press, 2016. 2

[36] Yamin Sepehri, Pedram Pad, Clment Kndig, Pascal Frossard, and L. Andrea Dunbar. Privacy-preserving image acquisition for neural vision systems. *IEEE Transactions on Multimedia*, pages 1–12, 2022. 2, 3

[37] Wanxin Shi, Zheng Huang, Honghao Huang, Chengyang Hu, Minghua Chen, Sigang Yang, and Hongwei Chen. Loen: Lensless opto-electronic neural network empowered machine vision. *Light: Science & Applications*, 11(1):1–12, 2022. 1, 2

[38] P.Y. Simard, D. Steinkraus, and J.C. Platt. Best practices for convolutional neural networks applied to visual document analysis. In *Seventh International Conference on Document Analysis and Recognition, 2003. Proceedings.*, pages 958–963, 2003. 18

[39] Matthieu Simeoni and Pol del Aguila Pla. matthieumeo/pycsou: Pycsou 1.0.6, Apr. 2021. 26

[40] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014. 5, 18

[41] Vincent Sitzmann, Steven Diamond, Yifan Peng, Xiong Dun, Stephen Boyd, Wolfgang Heidrich, Felix Heide, and Gordon Wetzstein. End-to-end optimization of optics and image processing for achromatic extended depth of field and super-resolution imaging. *ACM Trans. Graph.*, 37(4), jul 2018. 12, 14, 15, 16

[42] Jasper Tan, Salman S. Khan, Vivek Boominathan, Jeffrey Byrne, Richard Baraniuk, Kaushik Mitra, and Ashok Veeraraghavan. Canopic: Pre-digital privacy-enhancing encodings for computer vision. In *2020 IEEE International Conference on Multimedia and Expo (ICME)*, pages 1–6, 2020. 1, 2, 3

[43] Jasper Tan, Li Niu, Jesse K. Adams, Vivek Boominathan, Jacob T. Robinson, Richard G. Baraniuk, and Ashok Veeraraghavan. Face detection and verification using lensless cameras. *IEEE Transactions on Computational Imaging*, 5(2):180–194, 2019. 4

[44] Tianyu Wang, Shi-Yuan Ma, Logan G Wright, Tatsuhiro Onodera, Brian C Richard, and Peter L McMahon. An optical neural network using less than 1 photon per multiplication. *Nature Communications*, 13(1):1–8, 2022. 2

[45] Zihao W Wang, Vibhav Vineet, Francesco Pittaluga, Sudipta N Sinha, Oliver Cossairt, and Sing Bing Kang. Privacy-preserving action recognition using coded aperture videos. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pages 0–0, 2019. 1, 2, 3

[46] Gordon Wetzstein, Aydogan Ozcan, Sylvain Gigan, Shanhui Fan, Dirk Englund, Marin Soljačić, Cornelia Denz, David AB Miller, and Demetri Psaltis. Inference in artificial intelligence with deep optics and photonics. *Nature*, 588(7836):39–47, 2020. 2

[47] Logan G Wright, Tatsuhiro Onodera, Martin M Stein, Tianyu Wang, Darren T Schachter, Zoey Hu, and Peter L McMahon. Deep physical neural networks trained with backpropagation. *Nature*, 601(7894):549–555, 2022. 15

[48] Zhenyu Wu, Haotao Wang, Zhaowen Wang, Hailin Jin, and Zhangyang Wang. Privacy-preserving deep action recognition: An adversarial learning framework and a new dataset. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(4):2126–2139, 2022. 3

[49] Matthew D. Zeiler, Dilip Krishnan, Graham W. Taylor, and Rob Fergus. Deconvolutional networks. In *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 2528–2535, 2010. 26

[50] Huan Zhang, Hongge Chen, Zhao Song, Duane Boning, Inderjit dhillon, and Cho-Jui Hsieh. The Limitations of Adversarial Training and the Blind-Spot Attack. In *7th International Conference on Learning Representations, {ICLR} 2019, New Orleans, LA, USA, May 6-9, 2019*, 2019. 3

[51] Tiankuang Zhou, Lu Fang, Tao Yan, Jiamin Wu, Yipeng Li, Jingtao Fan, Huaqiang Wu, Xing Lin, and Qionghai Dai. In situ optical backpropagation training of diffractive optical neural networks. *Photon. Res.*, 8(6):940–953, Jun 2020. 15

[52] A. Zomet and S.K. Nayar. Lensless imaging with a controllable aperture. In *2006 IEEE Computer Society Conference*

on Computer Vision and Pattern Recognition (CVPR'06), volume 1, pages 339–346, 2006. 2

A. Digital twin modeling

Fig. A.1a illustrates our physical setup: the scene of interest is at a fixed distance d_1 from the optical encoder, which itself is at a distance d_2 from the image plane.

A.1. Convolutional relationship

We adopt a common assumption from *scalar diffraction theory*, namely that image formation is a linear shift-invariant (LSI) system [13]. This implies that there exists an impulse response, *i.e*. a point spread function (PSF), that can be convolved with the input scene to obtain the output image. In Fourier optics, this convolutional relationship is between the *scaled* scene and its image at a distance z, namely

$$U_2(\boldsymbol{x}; z, \lambda) = \int_{\mathbb{R}^2} d\boldsymbol{r}\, h(\boldsymbol{x} - \boldsymbol{r}; z, \lambda) \left[\frac{1}{|M|} U_0\Big(\frac{\boldsymbol{r}}{M}; \lambda\Big) \right], \tag{A.1}$$

where U_0 and U_2 are the wave fields, *i.e*. complex amplitudes, at the scene and image planes respectively, $\boldsymbol{x} \in \mathbb{R}^2$ and $\boldsymbol{r} \in \mathbb{R}^2$ are coordinates on the image and scene planes respectively, h is the PSF, and $M = -d_2/d_1$ is a magnification factor that also accounts for inversion [13]. Note that this convolution is dependent on the wavelength λ.

This LSI assumption significantly reduces the computational load for simulating optical wave propagation, as the convolution theorem and the fast Fourier transform (FFT) algorithm can be used to efficiently evaluate the wave field at the image plane via the spatial frequency domain. With the scaled Fourier transform, or equivalently chirp Z-transform, one can overcome the sampling limitations of the FFT and obtain the wave field at arbitrary resolutions [3,29]. An aperture, or some form of cropping, helps to enforce the LSI assumption in order to avoid new patterns from emerging at the sensor for lateral shifts at the scene plane. This is particularly necessary for encoders with a large support, *e.g*. those of lensless cameras.

For lenses, the PSF in Eq. (A.1) can be approximated by the Fraunhofer diffraction pattern (scaled Fourier transform) of the aperture function. For an arbitrary mask, the simplifications resulting from a lens are not possible and a more exact diffraction model is needed to predict the image pattern, *e.g*. Fresnel propagation or the angular spectrum method [13]. We employ the bandlimited angular spectrum method (BLAS) which produces accurate simulations for both near- and far-field [28].

Incoherent, polychromatic illumination. The convolutional relationship in Eq. (A.1) is for coherent illumination, *e.g*. coming from a laser. However, illumination from natural scenes typically consists of diffuse or extended sources which are considered to be *incoherent*. In such cases, impulses at the image plane vary in a statistically independent fashion, thus requiring them to be added on an intensity basis [13]. In order words, for incoherent illumination, the convolution of Eq. (A.1) should be expressed with respect to intensity:

$$I_2(\boldsymbol{x}; z, \lambda) = \int_{\mathbb{R}^2} d\boldsymbol{r}\, p(\boldsymbol{x} - \boldsymbol{r}; \lambda) \left[\frac{1}{|M|^2} I_0\Big(\frac{\boldsymbol{r}}{M}; \lambda\Big) \right], \tag{A.2}$$

where I_0 and I_2 are the intensities at the scene and image planes respectively, with image intensity defined as the average instantaneous intensity:

$$I(\boldsymbol{x}; z, \lambda) = \mathbb{E}\big[|U(\boldsymbol{x}; z, \lambda, t)|^2\big], \tag{A.3}$$

and the intensity PSF is equal to the squared magnitude of the amplitude PSF [13], *i.e*. of Eq. (A.1):

$$p(\boldsymbol{x}; z, \lambda) = |h(\boldsymbol{x}; z, \lambda)|^2. \tag{A.4}$$

For polychromatic simulation, each wavelength has to be simulated independently. Converting this multispectral data to RGB is typically done in two steps: (1) mapping each wavelength to the XYZ coordinates defined by the International Commission on Illumination[7] and (2) converting to red-green-blue (RGB) values based on a reference white.[8]

A.2. Point spread function modeling of a mask-based encoder

We model a programmable mask as a superposition of apertures for each adjustable pixel:

$$M(\boldsymbol{x}) = \sum_{k}^{K} w_k A(\boldsymbol{x} - \boldsymbol{x}_k), \tag{A.5}$$

where the complex-valued weights $\{w_k\}_{k=1}^K$ satisfy $|w_k| \leq 1$, the coordinates $\{(\boldsymbol{x}_k)\}_{k=1}^K$ are the centers of the mask pixels, and the aperture function $A(\boldsymbol{x})$ is assumed to be identical for each pixel. The weights $\{w_k\}_{k=1}^K$ correspond to the mask pattern we would like to determine for a given task.

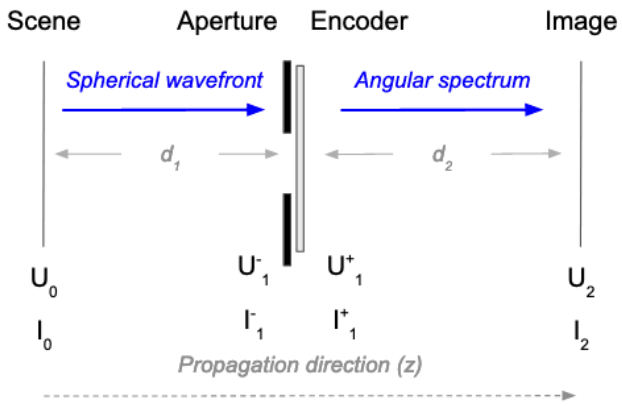
Our modeling of the PSF for propagation *through* the mask is similar to that of [41], namely for each wavelength λ, we simulate the propagation (see Fig. A.1a for the wave field variable at different planes):

1. *From the scene to the optical element*: propagation is modeled by spherical wavefronts. Assuming a point source at the scene plane U_0, we have the following wave field at the aperture plane:

$$U_1^-(\boldsymbol{x}; z = d_1, \lambda) = \exp\Big(j \frac{2\pi}{\lambda} \sqrt{\|\boldsymbol{x}\|_2^2 + d_1^2}\Big), \tag{A.6}$$

[7] https://en.wikipedia.org/wiki/CIE_1931_color_space

[8] http://www.brucelindbloom.com/index.html?Eqn_RGB_XYZ_Matrix.html



(a)

(b)

Figure A.1. (a) Propagation setup. Not drawn to scale for visualization purposes. (b) Example physical measurement setup.

where d_1 is the distance between the scene and the camera aperture.

2. *At the optical element*: the wave field is multiplied with the mask pattern:

$$U_1^+(\boldsymbol{x}; z = d_1, \lambda) = U_1^-(\boldsymbol{x}; z = d_1, \lambda) M(\boldsymbol{x}). \quad \text{(A.7)}$$

Note that an infinitesimally small distance is assumed between the opening of the aperture U_1^- and the exit of the mask U_1^+.

3. *To the sensor*: free-space propagation according to scalar diffraction theory [13] as light is diffracted by the optical element. This yields the following wave field at the sensor plane:

$$U_2(\boldsymbol{x}; z = d_1 + d_2, \lambda) = \mathcal{F}^{-1}\Big(\mathcal{F}\big(U_1^+(\boldsymbol{x}; z = d_1, \lambda)\big) H(\boldsymbol{u}; z = d_2, \lambda)\Big), \quad \text{(A.8)}$$

where $\mathcal{F}$ and $\mathcal{F}^{-1}$ denote the spatial Fourier transform and its inverse, $\boldsymbol{u} \in \mathbb{R}^2$ are spatial frequencies of $\boldsymbol{x}$, and the free-space frequency response according BLAS is given by:

$$H(\boldsymbol{u}; z = d_2, \lambda) = e^{j\frac{2\pi}{\lambda} d_2 \sqrt{1 - \|\lambda \boldsymbol{u}\|_2^2}} \operatorname{rect2d}\Big(\frac{\boldsymbol{u}}{2\boldsymbol{u}_{\text{limit}}}\Big), \quad \text{(A.9)}$$

where rect2d is a 2D rectangular function for bandlimiting the frequency response and the bandlimiting frequencies are given by

$$\boldsymbol{u}_{\text{limit}} = \frac{\sqrt{(d_2/\boldsymbol{S})^2 + 1}}{\lambda}, \quad \text{(A.10)}$$

where $\boldsymbol{S} \in \mathbb{R}^2$ are the physical dimensions of the propagation region, in our case the physical dimensions of the sensor.

4. *Intensity at sensor*: as we are simulating incoherent light, we require the squared modulus of the wave field:

$$p(\boldsymbol{x}; z = d_1 + d_2, \lambda) = \big|U_2(\boldsymbol{x}; z = d_1 + d_2, \lambda)\big|^2. \quad \text{(A.11)}$$

A.3. More on modeling a programmable mask

A key component in the above PSF simulation is modeling the complex-valued mask $M(\boldsymbol{x})$ associated with the programmable mask, which could be an LCD as in our setup or a spatial light modulator (SLM). Two assumptions are commonly made in its modeling:

- The mask is assumed to be either a phase transformation, *i.e.* $|M(\boldsymbol{x})| = 1$, or an amplitude transformation, *i.e.* $M(\boldsymbol{x}) \in [0, 1]$.

- The mask is discretized according to its resolution. This approximation neglects *deadspace* (or equivalently the fill factor) of individual pixels, namely the regions of the mask that are not programmable.

Our modeling of the programmable mask in Eq. (A.5) takes into account deadspace, whereas others simply discretize the mask according to its resolution [32]. However, this model assumes that no stray light passes between the pixels.

While numerical discretization may not be able to perfectly sample $M(\boldsymbol{x})$ to account for arbitrary shifts of $A(\cdot)$ in Eq. (A.5), these shifts can be accounted for in the spatial

frequency domain:

$$\mathcal{F}(M(\boldsymbol{x})) = M(\boldsymbol{u}) = A(\boldsymbol{u}) \sum_{k}^{K} w_k e^{j(\boldsymbol{u}\cdot\boldsymbol{x}_k)}. \quad \text{(A.12)}$$

Eq. (A.7) (for the wave field at the exit of the mask) can therefore be written as

$$U_1^+(\boldsymbol{x}; z = d_1, \lambda) = U_1^-(\boldsymbol{x}; z = d_1, \lambda)\mathcal{F}^{-1}(M(\boldsymbol{u})). \quad \text{(A.13)}$$

While this allows for arbitrary shifts, it requires an additional FFT and can be expensive when tracking gradients in order to optimize the mask weights w_k.

A cheaper way to account for deadspace is to discretize $M(\boldsymbol{x})$ at a finer resolution than of the mask, and only modulate those pixels which fall within the individual mask-pixel apertures (by setting the appropriate w_k value). While optimizing the programmable mask weights in our end-to-end approach, we adopt this latter simplification which is much more tractable when tracking gradients.

Other parameters that have been modeled for SLMs in holography and that are applicable to the context of imaging with programmable masks include: non-linear mapping between voltage-to-phase/amplitude of the individual mask pixels, Zernike coefficients to model deviations from theoretical propagation models, and content-dependent undiffracted (stray) light [32].

A.4. Specifics for the ST7735R LCD component and the Raspberry Pi High Quality camera

As the ST7735R component is originally intended to serve as a color display, it has an interleaved pattern of red, green, and blue filters as shown in Fig. A.2a,[9] which can be modeled as a wavelength-dependent version of Eq. (A.5)

$$M(\boldsymbol{x}; \lambda) = \sum_{c\in\{R,G,B\}} F_c(\lambda) \sum_{k_c}^{K_c} w_{k_c} A(\boldsymbol{x} - \boldsymbol{x}_{k_c}), \quad \text{(A.14)}$$

where:

- $\{F_c(\cdot)\}_{c\in\{R,G,B\}}$ is the wavelength-response of each color filter,
- $\left\{\{w_{k_c}\}_{k_c=1}^{K_c}, c \in \{R,G,B\}\right\}$ are real-valued weights for the red, green, and blue sub-pixels, and $\left\{\{\boldsymbol{x}_{k_c}\}_{k_c=1}^{K_c}, c \in \{R,G,B\}\right\}$ are their respective centers.

The display has a resolution of (128×160) color pixels, with three sub-pixels per color pixel as shown in Fig. A.2a. The dimension of each sub-pixel is $(0.06\,\text{mm} \times 0.18\,\text{mm})$ and the dimension of the entire screen is $(28.03\,\text{mm} \times 35.04\,\text{mm})$.[10] If we assume a uniform spacing of sub-pixels, this corresponds to a pixel pitch of roughly $(0.073\,\text{mm} \times 0.22\,\text{mm})$ and a fill-factor of 82%, namely deadspace of 18% around each sub-pixel.

The Raspberry Pi High Quality Camera[11] uses the Sony IMX477R back-illuminated sensor which has the following specifications: (3040×4056) pixel resolution, 7.9mm sensor diagonal, and a pixel size of $(1.55\mu\text{m} \times 1.55\mu\text{m})$, which corresponds to full sensor dimensions of $(4.71\,\text{mm} \times 6.29\,\text{mm})$. As the display of the ST7735R component is larger than the sensor, we use a subset of its pixels that covers the sensor area. From the pixel pitch of the ST7735R component determined above $(0.073\,\text{mm} \times 0.22\,\text{mm})$, it can be concluded that the number of sub-pixels that overlap the sensor is around (64×22). Moreover, for enforcing the LSI assumption described in Appendix A.1, we crop the LCD such that 80% of sensor surface is exposed, which corresponds to $51 \times 22 = 1122$ LCD sub-pixels. This is the number of sub-pixels that we optimize in our experiments.

Table B.1. Dataset distribution and simulation parameters.

Dataset →	MNIST	CelebA	CIFAR10
Scene-to-encoder	40 cm	55 cm	40 cm
Object height	12 cm	27 cm	25 cm
Signal-to-noise ratio	40 dB	40 dB	40 dB

B. Simulating an image of the desired scene

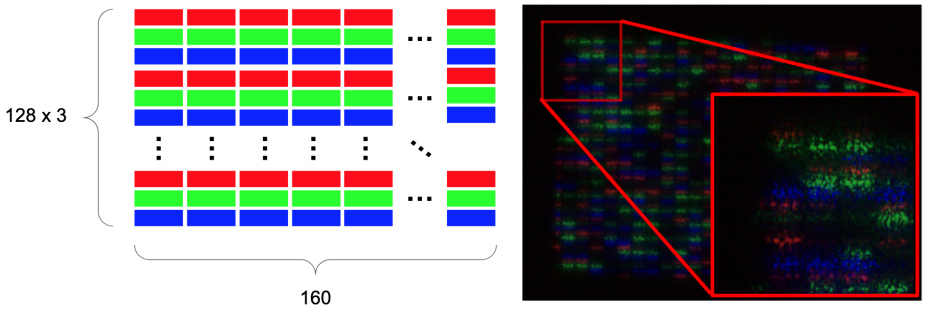
When simulating the propagation between two planes, as shown in Fig. A.1a, one may wish that the scene in the I_0 plane corresponds to the content of a digital image. This section describes how to pre-process an image such that its output corresponds to (1) a scene at a distance d_1 from the camera, (2) content from the original image having a height h_{obj} at the plane d_1 from the camera, and (3) a measurement taken by a camera of a known PSF. While Alg. 1 describes this simulation procedure succinctly, we dive into more detail in this section.

An RGB image can be interpreted as image intensities at three wavelengths: red, green, and blue [41]. We use the following wavelengths for red, green, and blue respectively: $640\,\text{nm}$, $550\,\text{nm}$, and $460\,\text{nm}$. For a grayscale image, such as images from MNIST [23], the same data can be used across channels, or it can be convolved with a grayscale version of the PSF.

[9] More information can be found on the device driver datasheet: https://cdn-shop.adafruit.com/datasheets/ST7735R_V0.2.pdf

[10] ST7735R breakout board datasheet: https://cdn-shop.adafruit.com/datasheets/JD-T1800.pdf

[11] Raspberry Pi High Quality Camera datasheet: https://cdn-shop.adafruit.com/product-files/4561/4561+Raspberry+Pi+HQ+Camera+Product+Brief.pdf



(a) (b)

Figure A.2. Visualizing pixel layout of the ST7735R component. (a) Red, green, blue color filter arrangement. (b) Zooming into section of a measured point spread function for a random pattern.

Given an image $\boldsymbol{x} \in \mathbb{R}^{C\times H\times W}$ with C channels and an intensity PSF $\boldsymbol{p} \in \mathbb{R}^{C\times H_{\text{PSF}}\times W_{\text{PSF}}}$, the simulation of a sensor measurement $\boldsymbol{v} \in \mathbb{R}^{C\times DH_{\text{PSF}}\times DW_{\text{PSF}}}$ (with an optional downsampling factor $D \geq 1$) can be summarized by the following steps:

1. Resize $\boldsymbol{x}$ to the PSF's dimension to obtain $\boldsymbol{x}_{\text{scene}} \in \mathbb{R}^{C\times H_{\text{PSF}}\times W_{\text{PSF}}}$, while preserving $\boldsymbol{x}$'s original aspect ratio and scaling it to correspond to a desired object height (or width). The details of this rescaling are explained in Appendix B.1.

2. Convolve each channel of $\boldsymbol{x}_{\text{scene}}$ with the corresponding PSF channel to obtain $\boldsymbol{v} \in \mathbb{R}^{C\times H_{\text{PSF}}\times W_{\text{PSF}}}$:

$$\boldsymbol{v}[c] = \boldsymbol{x}_{\text{scene}}[c] * \boldsymbol{p}[c], \quad c = 1, 2, \ldots, C. \tag{B.15}$$

   Due to large convolution kernels, this is typically best done in the spatial frequency domain, where convolution corresponds to an element-wise multiplication, and the FFT algorithm can be used to efficiently project between spatial and spatial frequency domains.

3. If $D > 1$, downsample the convolution output to the sensor resolution. We apply bilinear interpolation for this resizing.

$$\boldsymbol{v}[c] = \text{Downsample}(\boldsymbol{v}[c], D), \quad c = 1, 2, \ldots, C. \tag{B.16}$$

4. Add noise at a desired signal-to-noise ratio (SNR). More on this in Appendix B.2.

$$\boldsymbol{v}[c] = \boldsymbol{v}[c] + \boldsymbol{n}[c], \quad c = 1, 2, \ldots, C. \tag{B.17}$$

Having a faithful estimate of the intensity PSF $\boldsymbol{p}$ is the most vital part of the above simulation. For a fixed optical encoder, if a physical setup is available, the best approach may be to simply measure the PSF by placing a point source (*i.e.* a white LED behind a pinhole as shown in Fig. C.3c) at the desired distance and taking the resulting measurement as the intensity PSF. Some post-processing may be necessary to remove sensor noise and balance color channels. For encoders that have no parametric function, such as pseudo-random diffusers [1], measuring the PSF may be the only viable option.

For encoders that have a parametric function, *e.g.* lenses or programmable masks, it is possible to simulate the intensity PSF. This is in fact necessary for most end-to-end optimization techniques, unless forward-/back-propagation are done directly with hardware [51]. Even for a parametric encoder, it can be useful to measure the PSF (if a physical setup is available) in order to reduce mismatch due to model assumptions / simplifications [6]. In Appendix A.2, we describe our modeling of the PSF of a programmable mask at a particular wavelength, and explain how we account for the specifics of the ST7735R component and the Raspberry Pi High Quality Camera in our proposed system.

Using physical measurements. The above simulation of Fig. A.1a seeks to replicate the physical measurement setup shown in Fig. A.1b, namely projecting the image of the desired scene on a display at a distance d_1 from the camera. While such a measurement would produce more realistic results, it can be very time-consuming for an *entire dataset* of images. If this dataset is to be used for a task with a fixed optical encoder, a lens or diffuser, it may be worth the time and effort as the measurement only has to be done once. For a task that seeks to optimize the optical encoder in an end-to-end fashion, new measurements would have to be performed during training whenever updates are made to the optical encoder. This is highly impracticable for optical encoders that require precise fabrication [6, 27, 41]. In the case of programmable optical encoders, alternating between physical measurements and updating the optical encoder lends itself to hardware-in-the-loop / physics-aware training [32, 47]. This has the potential to reduce model-mismatch but at the cost of longer training, due to acquisi-

tion time and a lack of parallelization.

B.1. Rescaling image to PSF resolution for a desired object height

Note that in this section we use capital letters to denote dimensions in pixels or scalar factors, and lower case letters for dimensions in meters. The goal of this step in the simulation of Appendix B is to rescale a digital image $\boldsymbol{x} \in \mathbb{R}^{C\times H\times W}$ such that its convolution with a digital PSF $\boldsymbol{p} \in \mathbb{R}^{C\times H_{\text{PSF}}\times W_{\text{PSF}}}$ corresponds to the setup in Fig. A.1a for an object of height h_{obj} at the scene plane. For such a configuration, namely a scene-to-encoder distance of d_1 and an encoder-to-image distance of d_2, the object height *at the sensor* is given by:

$$h_{\text{sensor}} = h_{\text{obj}}(d_2/d_1) = h_{\text{obj}} \cdot |M|. \tag{B.18}$$

For our simulation we are interested in the number of pixels that this height corresponds to. If our PSF was measured for the above distances with a sensor resolution of $(H_{\text{sensor}} \times W_{\text{sensor}})$ pixels and a pixel pitch of p, the sensor will have captured a PSF for a scene of the following physical dimensions:

$$(h_{\text{scene}} \times w_{\text{scene}}) = \Big(\frac{pH_{\text{sensor}}}{|M|} \times \frac{pW_{\text{sensor}}}{|M|}\Big). \tag{B.19}$$

Consequently, the object height *in pixels* at the sensor is approximately given by:

$$H_{\text{pixel}} = \text{round}\Big(\frac{h_{\text{obj}} H_{\text{PSF}}}{h_{\text{scene}}}\Big). \tag{B.20}$$

Therefore, to rescale the original input image $\boldsymbol{x} \in \mathbb{R}^{C\times H\times W}$ to the PSF resolution, while preserving its aspect ratio and scaling it such that it corresponds to the desired object height, we need to perform the following steps:

1. Resize $\boldsymbol{x}$ to $\big(C \times \text{round}(SH) \times \text{round}(SW)\big)$ where $S = (H_{\text{pixel}}/H)$.
2. Pad above to $(C \times H_{\text{PSF}} \times W_{\text{PSF}})$.

The resulting image $\boldsymbol{x}_{\text{scene}} \in \mathbb{R}^{C\times H_{\text{PSF}}\times W_{\text{PSF}}}$ can then be convolved with the PSF to simulate a propagation as in Fig. A.1a.

B.2. Adding noise at a desired signal-to-noise ratio

Different types of noise can be added during simulation. In practice, read noise at a sensor follows a Poisson distribution with respect to the input. However, as this distribution is not differentiable with respect to the optical encoder parameters, Gaussian noise is used instead [27, 41].

In order to add generated noise to a signal and obtain a desired signal-to-noise ratio (SNR), the generated noise must be scaled appropriately. SNR (in dB) is defined as:

$$\text{SNR} = 10\log_{10}(\sigma_S^2/\sigma_N^2), \tag{B.21}$$

where σ_S^2 is the clean image variance and σ_N^2 is the generated noise variance. For a target SNR T, the generated noise can be scaled with the following factor:

$$k = \sqrt{\frac{\sigma_S^2}{\sigma_N^2 10^{(T/10)}}}. \tag{B.22}$$

In our simulation, we generate noise following a Poisson distribution. As we do not backpropagate through noise generation to the optical encoder parameters, we opt for this more realistic realization of noise.

C. Baseline and proposed cameras

As mentioned in Appendix B, a faithful estimate of the intensity PSF is an important part of the simulation for our baseline and proposed cameras: *Lens*, *Coded aperture* [2], *Diffuser* [1], *Fixed mask (m)*, *Fixed mask (s)*, and *Learned mask*. For our experiments in Section 5, we use the Raspberry Pi High Quality Camera for both the PSF measurement and in simulating the PSF. Measured PSFs are obtained by placing a white LED behind a pinhole aperture, as shown in Fig. C.3c, at the target distance, and measuring the response in an environment with no external light. Simulated PSFs are obtained with the approach described in Appendix A.2. Except for *Lens*, the scene, encoder, and image planes (U_0, U_1^+, U_2 respectively in Fig. A.1a) for simulating the PSFs take on the size and resolution of the Raspberry Pi High Quality Camera downsampled by a factor of $D = 8$, namely a pixel resolution of (380×507) and a pixel size of $(12.4\mu\text{m} \times 12.4\mu\text{m})$. Due to the very compact support of its PSF, we use the full resolution of the Raspberry Pi High Quality Camera, namely (3040×4056).

Below are technical details regarding each PSF:

- *Lens*: measured PSF for the camera shown in Fig. C.3a, which has a 6mm wide angle lens[12] focused at 40 cm. The lens and its objective have a thickness of 34 mm, and the lens is 7.53 mm from the sensor.
- *Coded aperture*: a binary mask is generated by (1) generating a length-63 maximum length sequence (MLS) binary array,[13] (2) repeating the sequence to create a 126-length sequence, and (3) computing the outer product with itself to create a 126×126 matrix. These are the same steps for generating a coded aperture mask as for FlatCam [2], except that we use a shorter MLS sequence (63 instead of 255) to obtain a feature size of 30 µm (as in [2]). The mask covers 80% of the

[12] 6mm Wide Angle Lens for Raspberry Pi HQ Camera datasheet: https://cdn-shop.adafruit.com/product-files/4563/4563-datasheet.pdf

[13] Using the SciPy function `max_len_seq`: https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.max_len_seq.html

sensor height (as *Fixed SLM (s)* and *Learned SLM* below). For the PSF, we simulate the mask's diffraction pattern for a distance of $d_2 = 0.5\,\mathrm{mm}$, matching the distance in [2].

- *Diffuser*: measured PSF for the camera shown in Fig. C.3b, where the diffuser is placed roughly $4\,\mathrm{mm}$ from the sensor. The diffuser is double-sided tape as in the DiffuserCam tutorial [5]. In [4], the authors demonstrate the effectiveness of this simple diffuser when used with the Raspberry Pi High Quality Camera. It is less than $1\,\mathrm{mm}$ thick and is placed roughly $4\,\mathrm{mm}$ from the sensor.
- *Fixed mask (m)*: measured PSF for the proposed camera shown in Fig. 3 for a random pattern (uniform distribution). With a stepper motor, the mask-to-sensor distance is programmatically set to $4\,\mathrm{mm}$ to match the distance of the diffuser-based camera.
- *Fixed mask (s)*: simulated PSF for the proposed camera, using the approach described in Appendix A.2 for a random set of amplitude values (uniform distribution) and a mask-to-sensor distance of $4\,\mathrm{mm}$. The aperture is set such that mask pixels covering 80% of the sensor surface area are exposed. This corresponds to $51 \times 22 = 1122$ sub-pixels as described in Appendix A.4.
- *Learned mask*: simulated PSF for the proposed camera that is obtained by optimizing Eq. (1) for the mask weights, and simulating the corresponding PSF with the approach described in Appendix A.2 for a mask-to-sensor distance of $4\,\mathrm{mm}$. Like *Fixed mask (s)*, pixels that cover 80% of the sensor surface area are used, corresponding to $51 \times 22 = 1122$ pixels. As the mask values are updated after backpropagation during training, the resulting PSF is different for each batch.

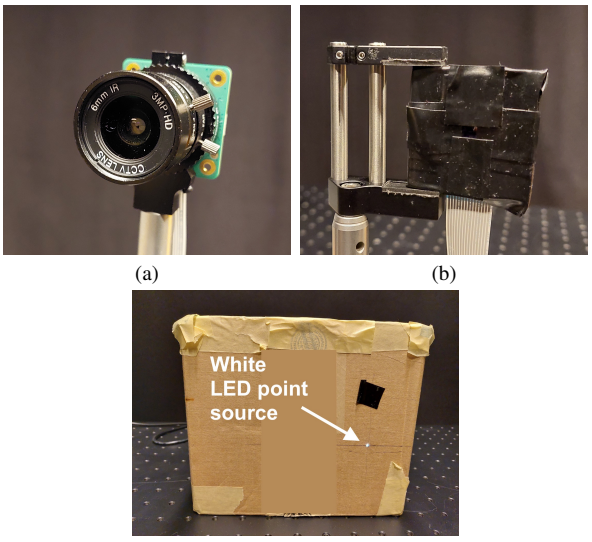
D. Training details and classification architectures

All experiments in Section 5 were run on a Dell Precision 5820 Tower X-Series (08B1) machine with an Intel i9-10900X 3.70 GHz CPU and two NVIDIA RTX A5000 GPUs. PyTorch [31] was used for dataset preparation and training.

D.1. Loss functions

For the multi-label classification tasks (MNIST and CIFAR10), we use a cross entropy loss in optimizing Eq. (1):

$$\mathcal{L}(\boldsymbol{y}, \hat{\boldsymbol{y}}) = -\sum_{c=1}^{C} \log \frac{\exp(\hat{\boldsymbol{y}}_c)}{\sum_{i=1}^{C} \exp(\hat{\boldsymbol{y}}_i)} \boldsymbol{y}_c, \tag{D.23}$$



(a) (b)

(c)

Figure C.3. Baseline cameras: (a) lensed and (b) diffuser-based. (c) Light source (white LED behind a hole in cardboard box) for measuring the point spread function.

where $\boldsymbol{y} \in \mathbb{R}^C$ are one-hot encoded vectors corresponding to the ground-truth labels, and $\hat{\boldsymbol{y}} = D_{\boldsymbol{\theta}_D}\big(O_{\boldsymbol{\theta}_E}(\boldsymbol{x})\big) \in \mathbb{R}^C$ are the predicted scores for a given input $\boldsymbol{x}$ that passes through the optical encoder $O_{\boldsymbol{\theta}_E}(\cdot)$ and the digital decoder $D_{\boldsymbol{\theta}_D}(\cdot)$. For both MNIST and CIFAR10, $C = 10$.

For the binary classification tasks (gender and smiling classification with CelebA), we use a binary cross entropy loss in optimizing Eq. (1):

$$\mathcal{L}(y, \hat{y}) = -\Big[y \log \hat{y} + (1-y) \log(1-\hat{y})\Big]. \tag{D.24}$$

D.2. Normalization

For the fixed optical encoders, the embeddings $\{\boldsymbol{v}_i\}_{i=1}^N$ that are inputted to the classifiers are pre-computed with the approach described in Appendix B. The resulting augmented dataset is normalized (according to the augmented training set statistics). For *Learned mask*, we apply batch normalization [17] and a ReLU activation to the sensor embedding prior to passing it to the classifier.

D.3. Classification architectures

The following classification architectures are used in the experiments of Section 5.

Logistic regression (LR). The classifier performs the following steps:

1. Flatten sensor embedding.
2. Fully connected linear layer to 10 classes if MNIST or CIFAR10, else a single unit for CelebA.

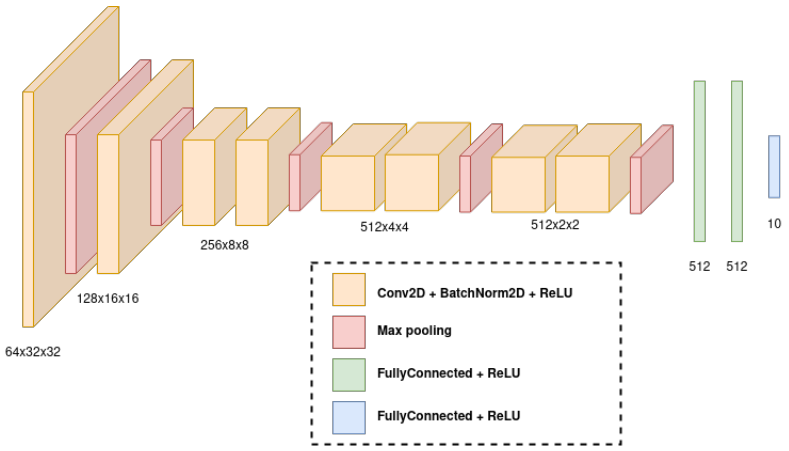


Figure D.4. VGG11 architecture for CIFAR10.

3. Softmax decision layer if MNIST or CIFAR10, else sigmoid activation for CelebA.

Two-layer fully connected neural network (FC). The classifier performs the following steps:

1. Flatten sensor embedding.
2. Fully connected linear layer to hidden layer of 800 units, as in [38].
3. Batch normalization.
4. ReLU activation.
5. Fully connected linear layer to 10 classes if MNIST or CIFAR10, else single unit for CelebA.
6. Softmax decision layer if MNIST or CIFAR10, else sigmoid activation for CelebA.

VGG11. The classifier performs the following steps:

1. Eight convolutional layers using (3×3) filters with max-pooling in between some layers.
2. Flatten.
3. Three-layer fully-connected network. The first two layers are preceded by $50\,\%$ dropout (in training) and followed by a ReLU activation.

The architecture can be seen in Fig. D.4, and is inspired by that of the original VGG paper [40]).[14]

D.4. Training hyperparameters

All classifiers are trained for 50 epochs. Unless noted otherwise, the Adam optimizer [21] is used with an initial learning rate of 0.001. Depending on the experiment, different batch sizes and learning rate schedules were used in order to avoid over-fitting. Appendices E to H describe the specific training hyperparameters for the end-to-end training experiments in Sections 5.1 to 5.3.

For *Learned mask* we tend to a larger batch size for speeding up training time, as the simulation of the new PSF (after updating the mask values) and of all the train / test examples results in a significantly larger training time. For the fixed encoders (the rest) the train and test examples can be simulated offline, *i.e.* as a pre-processing step. For our hardware and 50 epochs, it took around 20 min for training fixed encoders and 280 min for *Learned mask*.

E. MNIST – varying sensor dimension and architecture

Tab. E.2 details the training hyperparameters for the experiment in Section 5.1 that varies the embedding dimension and architecture for handwritten digit classification (MNIST). Train and test accuracy curves can be see in Fig. E.5. Example sensor embeddings can be found in Fig. E.6.

Table E.2. Training hyperparameters for experiment in Section 5.1 for handwritten digit classification (MNIST). LR and FC architectures are described in Appendix D.3 and Appendix D.3 respectively. *Schedule* denotes (if applied) after how many epochs the learning rate is reduced by a factor of 0.1. All models are trained for 50 epochs.

Configuration ↓	*Batch size*	*Schedule*
LR, fixed encoders	64	NA
LR, *Learned mask*	100	NA
FC, fixed encoders	32	20
FC, *Learned mask*	64	20

[14] Adapted code from this CIFAR10 architecture: https://github.com/kuangliu/pytorch-cifar/blob/master/models/vgg.py

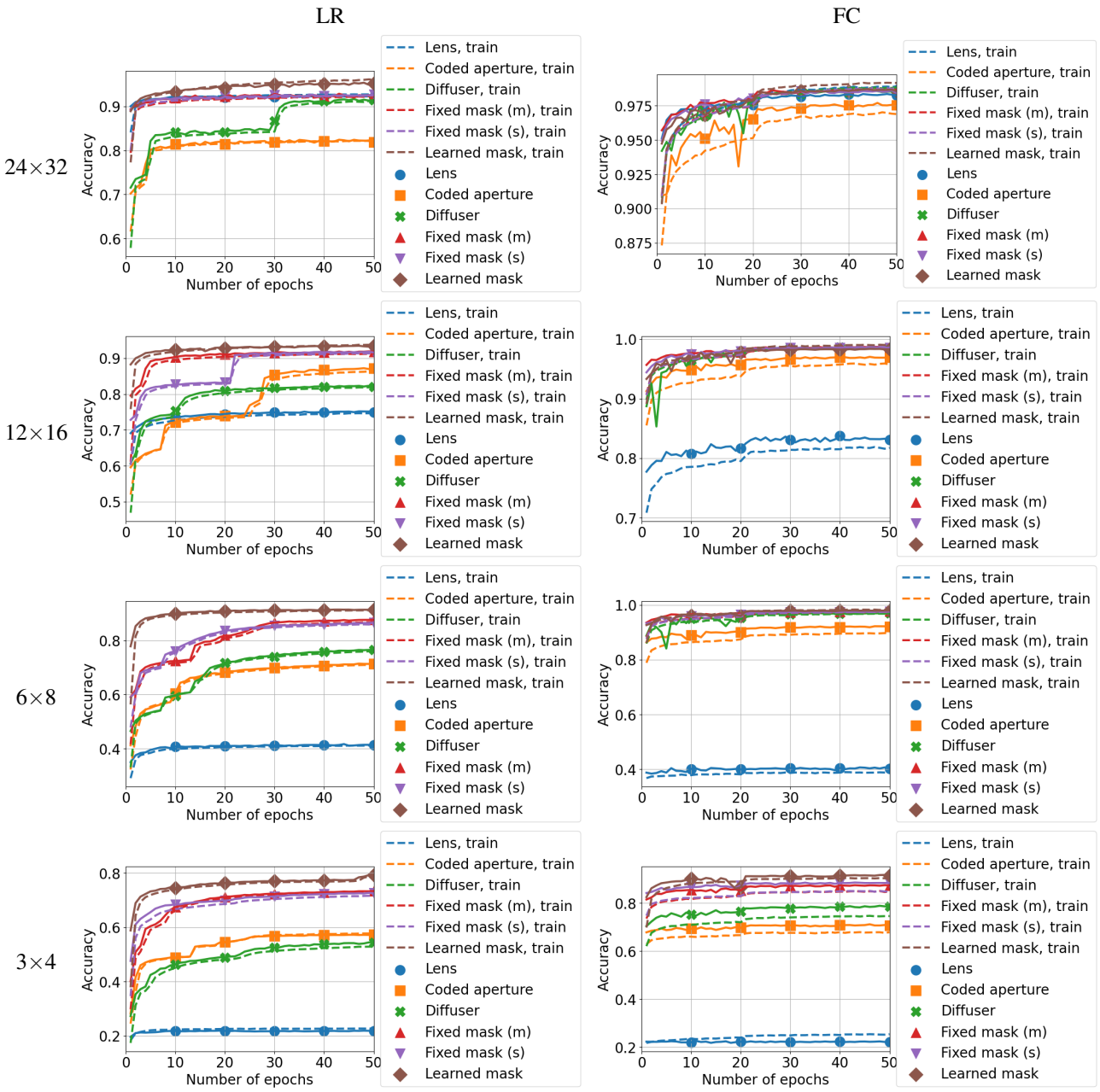


Figure E.5. Train and test curves for MNIST.

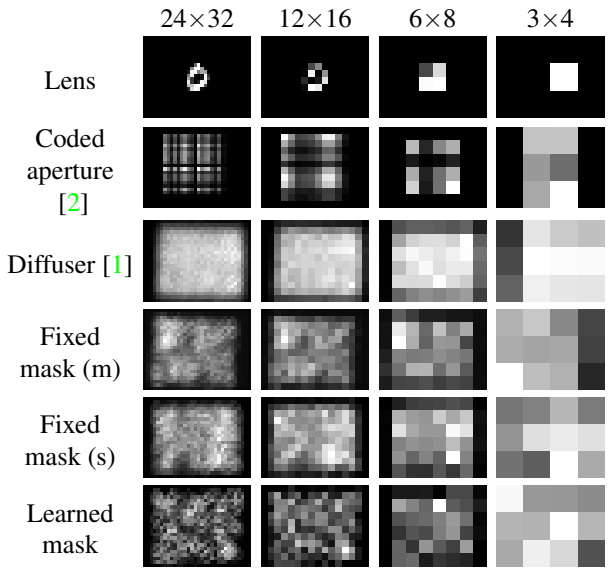


Figure E.6. Example sensor embeddings of the baseline and proposed camera systems.

F. MNIST – robustness to common image transformations

Only FC is used as a digital classifier in this experiment; the same hyperparameters shown in Tab. E.2 are used for the batch size and learning rate schedule. Train and test accuracy curves can be see in Fig. F.12. The classification accuracy under each random perturbation can be seen in Tab. F.3.

Table F.3. MNIST accuracy for two-layer fully connected network with 800 hidden units and an embedding dimension of (24×32) for a randomly transformed data.

Encoder ↓	Original	Shift	Rescale	Rotate	Perspective
Lens	*98.4%*	*86.4%*	*85.0%*	*95.1%*	*85.1%*
Coded aperture	97.7%	22.2%	82.8%	91.0%	31.3%
Diffuser	98.6%	31.8%	96.4%	95.7%	71.4%
Fixed mask (m)	**98.7%**	42.7%	96.7%	95.6%	80.6%
Fixed mask (s)	98.6%	37.9%	**96.8%**	95.5%	79.1%
Learned mask	**98.7%**	**68.8%**	92.5%	**95.8%**	**84.5%**

In the following subsections, we detail the distribution of perturbations applied to the train and test sets *offline*, *i.e.* as a pre-processing step when preparing the data for both fixed and learned encoders. Example sensor embeddings and reconstructions are shown to demonstrate how the different imaging systems cope with the random perturbations. For comparison, Fig. F.7 are example sensor embeddings and reconstructions when there are no perturbations (*Original*).

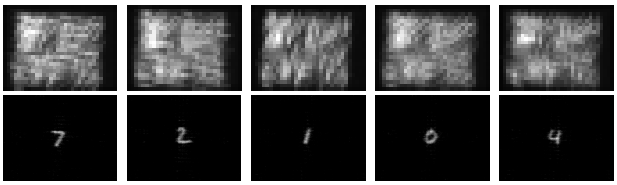


Figure F.7. Example sensor embeddings for *Fixed mask (m)* with no perturbations (top) and the corresponding reconstruction (bottom) using the convex optimization approach described in Appendix I.1.

F.1. Shift

While maintaining an object height of $12\,\mathrm{cm}$ and an object-to-camera distance of $40\,\mathrm{cm}$, shift the image in any direction along the object plane such that it is still fully captured by the sensor.

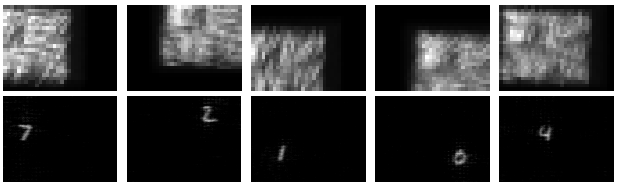


Figure F.8. Example sensor embeddings for *Fixed mask (m)* in the presence of shifting (top) and the corresponding reconstruction (bottom) using the convex optimization approach described in Appendix I.1.

F.2. Rescale

While maintaining an object-to-camera distance of $40\,\mathrm{cm}$, set a random height uniformly drawn from $[2\,\mathrm{cm}, 20\,\mathrm{cm}]$.

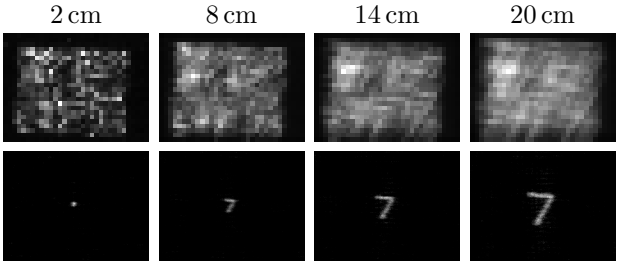


Figure F.9. Example sensor embeddings for *Fixed mask (m)* in the presence of rescaling (top) and the corresponding reconstruction (bottom) using the convex optimization approach described in Appendix I.1.

F.3. Rotate

While maintaining an object height of $12\,\mathrm{cm}$ and an object-to-camera distance of $40\,\mathrm{cm}$, uniformly draw a rotation angle from $[-90\,\mathrm{deg}, 90\,\mathrm{deg}]$.

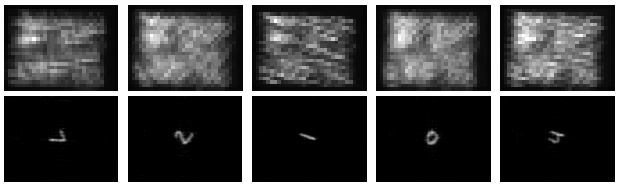


Figure F.10. Example sensor embeddings for *Fixed mask (m)* in the presence of random rotation (top) and the corresponding reconstruction (bottom) using the convex optimization approach described in Appendix I.1.

F.4. Perspective

While maintaining an object-to-camera distance of $40\,\mathrm{cm}$, perform a random perspective transformation via PyTorch's `RandomPerspective` with $100\,\%$ probability and a distortion factor of 0.5.[15]

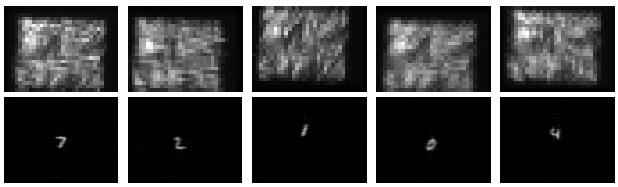


Figure F.11. Example sensor embeddings for *Fixed mask (m)* in the presence of random perspective distortions (top) and the corresponding reconstruction (bottom) using the convex optimization approach described in Appendix I.1.

G. CelebA – face attribute binary classification

Tab. G.4 details the training hyperparameters for the experiment in Section 5.2 that varies the embedding dimension for face attribute classification (CelebA). Example sensor embeddings can be found in Fig. G.13. Train and test accuracy curves can be see in Fig. G.14.

Table G.4. Training hyperparameters for experiment in Section 5.2 for face attribute classification (CelebA). The FC architecture – described in Appendix D.3 – was used for all imaging systems and embedding dimensions. *Schedule* denotes after how many epochs the learning rate is reduced by a factor of 0.1. All models are trained for 50 epochs.

Encoder ↓	*Batch size*	*Schedule*
Fixed encoders	64	10
Learned mask	64	20

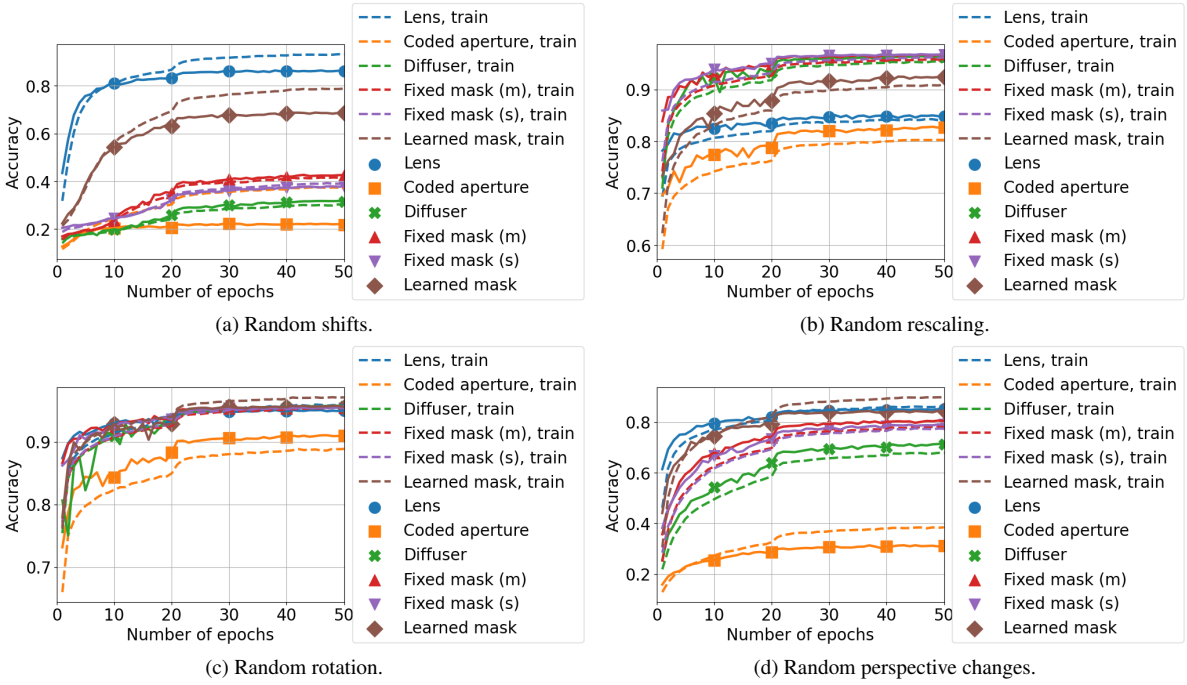
H. CIFAR10 - RGB object classification

Tab. H.5 details the training hyperparameters for the experiment in Section 5.3 that varies the embedding dimension for RGB object classification (CIFAR10). Stochastic gradient descent is used as an optimizer, with an initial learning rate of 0.01. We also apply online augmentation to minimize over-fitting, namely (1) random horizontal flipping and (2) padding and random cropping back to the original embedding dimensions (essentially random shifts). We apply different amount of padding depending on the embedding dimension and encoder, as shown in Tab. H.5. For all embedding dimensions, we resize the embedding to (32×32) as the VGG architecture typically operates on square inputs that can be downsampled at multiple steps until the classification layer.

Train and test accuracy curves can be see in Fig. H.15. Example sensor embeddings can be found in Fig. H.18. Note that the proposed system has a color filter, such that the measurements of *Fixed mask (m)*, *Fixed mask (s)*, and *Learned mask* do not resemble the "average" color of the scene. Fig. J.25 (bottom row) shows the PSFs of the learned masks for each embedding dimension. Fig. H.16 shows the color PSFs of *Fixed mask (m)* and *Fixed mask (s)*.

Fig. H.17 shows the confusion matrices for an embedding dimension of (26×37) for *Lens*, *Fixed mask (m)* (the best fixed lensless encoder), and *Learned mask*. For the lensless encoders, there is difficulty in distinguishing *cars-trucks*, *cats-dogs*, and *birds-deer*. The first two mistakes are understandable, the last ones less so. Perhaps it stems from the outdoor setting; although *deer-horse* would have been more understandable.

[15] `RandomPerspective` documentation: https://pytorch.org/vision/main/generated/torchvision.transforms.RandomPerspective.html



(a) Random shifts.

(b) Random rescaling.

(c) Random rotation.

(d) Random perspective changes.

Figure F.12. Train and test curves for MNIST simulated with perturbations.

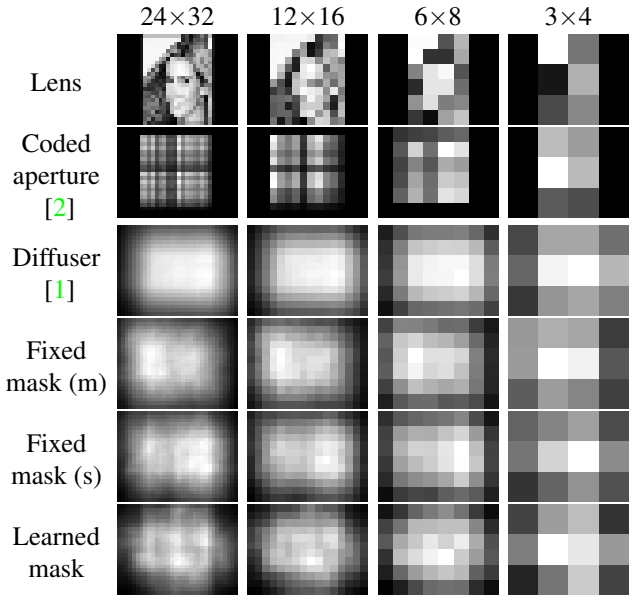
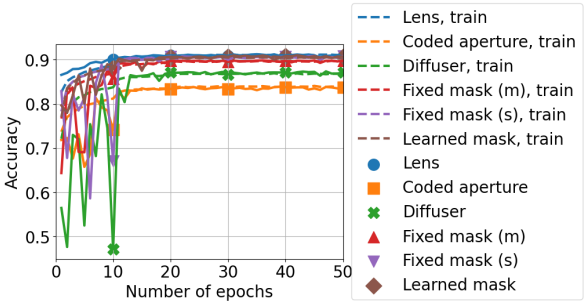


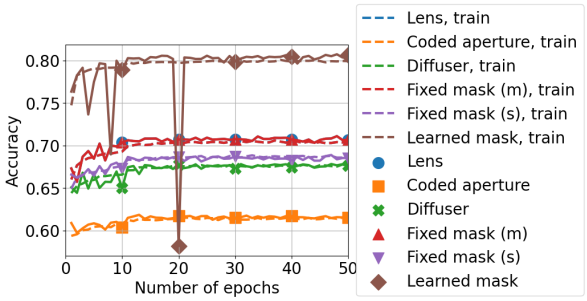
Figure G.13. Example sensor embeddings for CelebA.

Table H.5. Training hyperparameters for experiment in Section 5.3 for RGB object classification (CIFAR10). The VGG11 architecture – described in Appendix D.3 – was used for all imaging systems and embedding dimensions. *Schedule* denotes after how many epochs the learning rate is reduced by a factor of 0.1. All models are trained for 50 epochs. *Pad* denotes how much the input image was padded prior to random cropping for (3× 27×36), (3× 13×17), (3× 6×8), and (3× 3×4) respectively.

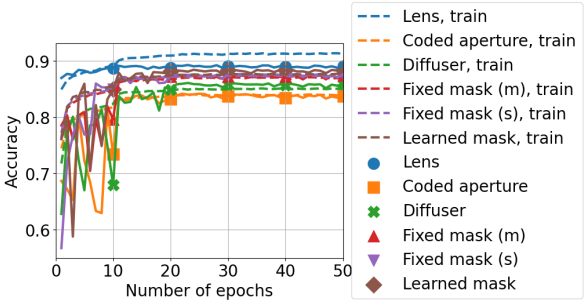
Encoder ↓	*Batch size*	*Schedule*	*Pad*
Lens	32	10	4-4-2-0
Fixed lensless	32	10	1-1-1-0
Learned mask	32	20	2-1-1-0



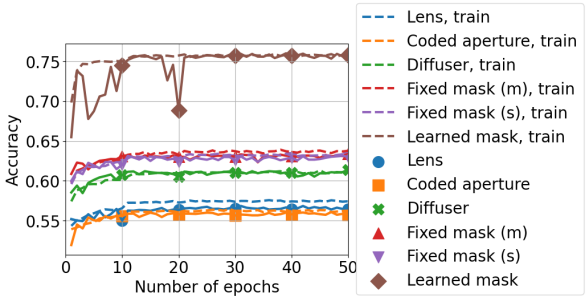
(a) Gender classification, 24×32.



(b) Gender classification, 3×4.

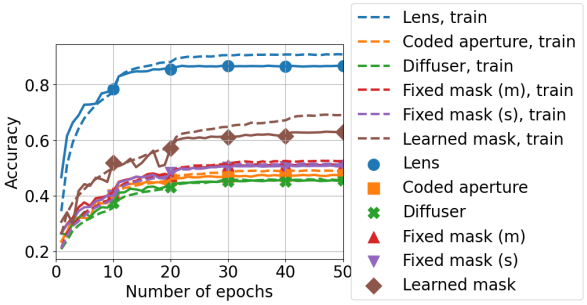


(c) Smiling classification, 24×32.

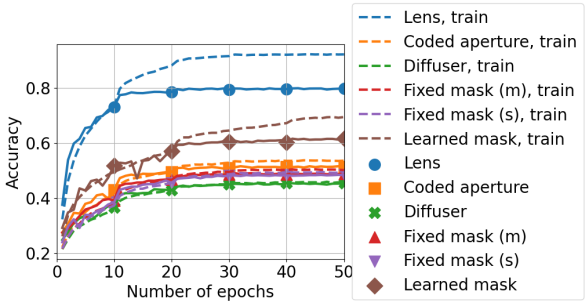


(d) Smiling classification, 3×4.

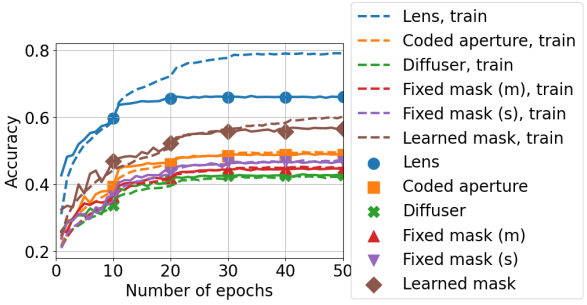
Figure G.14. Train and test curves for CelebA face attribute classification.



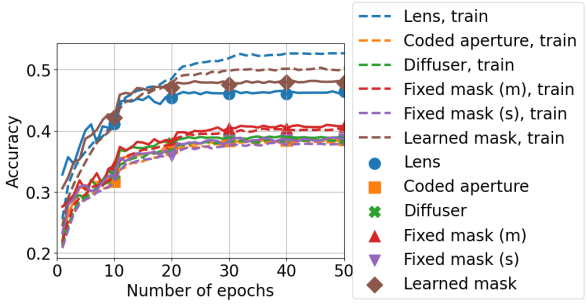
(a) 3×27×36.



(b) 3×13×17.

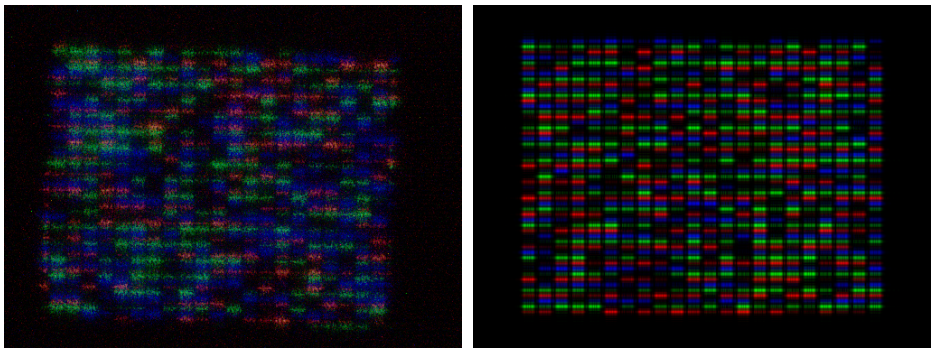


(c) 3×6×8.



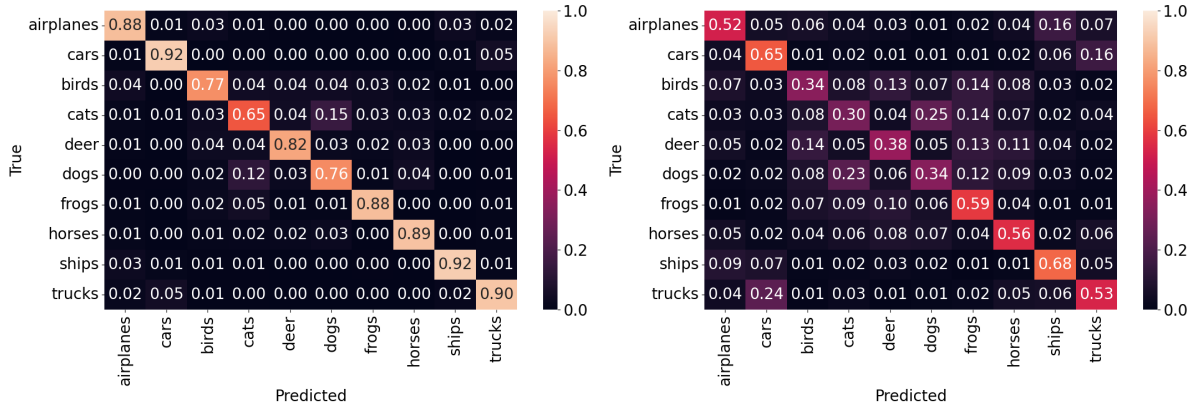
(d) 3×3×4.

Figure H.15. Train and test curves for CIFAR10 classification, varying embedding dimension.



(a) *Fixed mask (m).* (b) *Fixed mask (s).*

Figure H.16. RGB point spread functions of *Fixed mask (m)* and *Fixed mask (s)*. Note that different mask patterns are set.



(a) *Lens.* (b) *Fixed mask (m).*

(c) *Learned mask.*

Figure H.17. CIFAR10 confusion matrices for an embedding dimension of (27×36). See Tab. 3 (right side) for average accuracy.

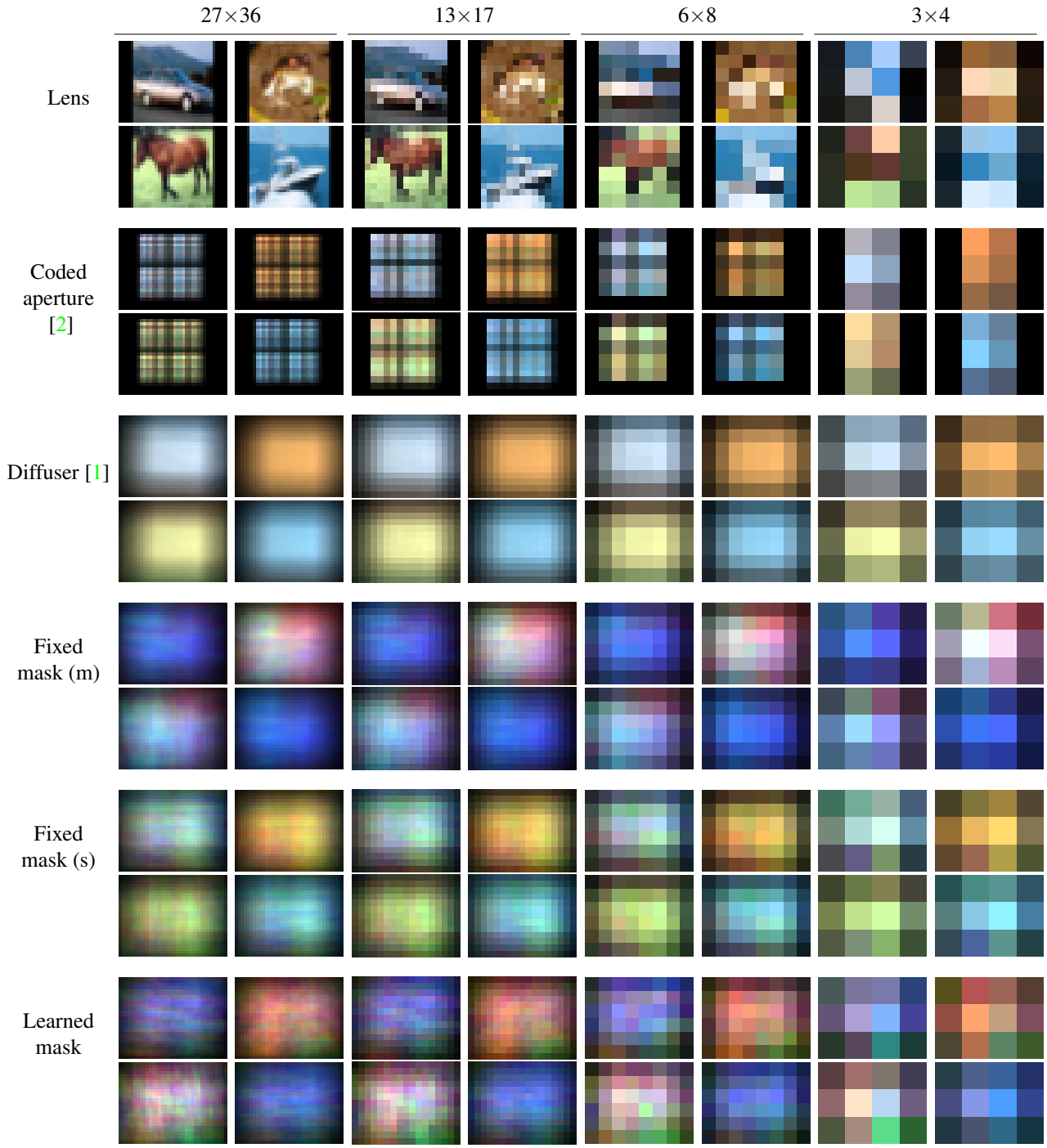


Figure H.18. Example sensor embeddings for CIFAR10; *car*, *frog*, *ship*, and *ship* examples (clockwise from top-left). Note that the proposed system has a color filter, such that the measurements of *Fixed mask (m)*, *Fixed mask (s)*, and *Learned mask* do not resemble the "average" color of the scene. Fig. J.25 (bottom row) shows the PSFs of the learned masks for each embedding dimension. Fig. H.16 shows the color PSFs of *Fixed mask (m)* and *Fixed mask (s)*.

I. Adversarial attacks

I.1. Convex optimization to recover object from downsampled lensless camera measurement

$$\hat{\boldsymbol{x}} = \underset{\boldsymbol{x}\geq 0}{\operatorname{argmin}} \frac{1}{2}\|\boldsymbol{y} - \boldsymbol{D}\boldsymbol{H}\boldsymbol{x}\|_2^2, \tag{I.25}$$

where $\boldsymbol{y}$ is the raw measurement, $\boldsymbol{D}$ is a downsampling operation whose output size corresponds to the sensor resolution, $\boldsymbol{H}$ is the high-resolution point spread function (PSF) of the system, $\boldsymbol{x}$ is the underlying object. This problem can be solved through projected gradient descent [9].

A common approach in inverse modeling is to add a prior to the loss. This also serves to regularize the solution. We tried adding the total variation (TV) norm *e.g.* $\|\Psi\boldsymbol{x}\|_1$ where Ψ is the finite-difference operator (as in [1]), but found results to be slightly worse (a bit blurrier) for non-sparse data such as faces.

For the first adversary experiment in Section 5.4 (example attacks in Fig. 5 and results in Tab. 4): we simulate examples with Alg. 1 to obtain several $\boldsymbol{y}$, and use the Pycsou library [39] to solve Eq. (I.25) numerically. For the correct PSF scenario, we use the same PSF used in simulating the example. For the bad PSF scenario, we randomly set the mask values and simulate a new PSF with the approach described in Section 4.3. This new PSF is used in solving Eq. (I.25).

I.2. Plaintext attack generator

In this section, we describe the training details for the second adversary experiment in Section 5.4 that attempts to reconstruct embedding dimensions of faces through a trained generator.

Dataset. We use the CelebA dataset [25] to train the decoder, namely a separate 100K files than were used to train the masks as in Section 5.2. All files are passed through the proposed imaging system with 1, 10, and 100 varying masks to produce 100K embeddings. When varying the number of plaintext attacks to produce the results in Tab. 5, a subset of these 100K embeddings are used to train the generator with a train-test split of 85-15.

Architecture. Fig. I.19 shows the architecture used for generating faces, which consists of:

1. Flatten sensor embedding.
2. Two-layer fully connected network with ReLU activations ($10'000$ and $19'008$ hidden units).
3. Reshaping output to a cube of dimension $(32 \times 27 \times 22)$.
4. Two 2D transposed convolution operators [49] – upsampling followed by convolution with a set of (3×3) filters – followed by 2D batch normalization and ReLU activation.
5. A final 2D transposed convolution operator with a sigmoid activation to restrict the output image pixels to $[0, 1]$.

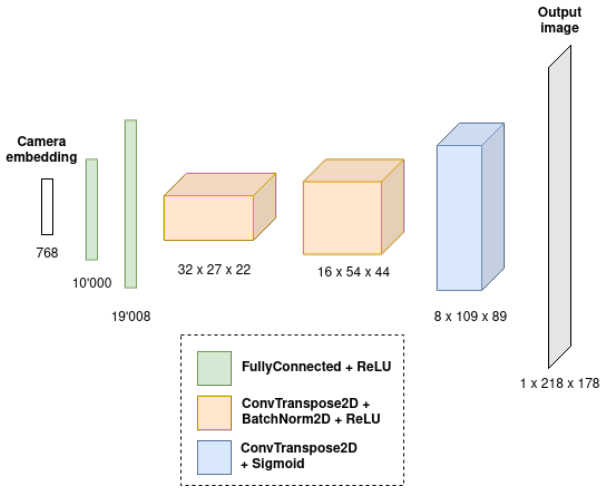


Figure I.19. Architecture for plaintext generator.

Our architecture is inspired by that of autoencoders and generative adversarial networks [19] (fully connected layers followed by convolutional layers).[16]

We also tried the following but they did not produce better results:

- Increasing the number of (ConvTranspose2D + BatchNorm2d + ReLU) layers.
- Training a fully-connected layer prior to a (frozen and unfrozen) pre-trained StyleGAN2 [20].

Training hyperparameters. Due to the significant difference in number of training examples, we use different batch sizes: 4 for the 100 plaintext attacks scenario, 16 for 1K, and 32 for the 10K and 100K. All models were training with stochastic gradient descent and a learning rate of 0.01.

PSNR and SSIM test curves can be seen in Fig. I.23, demonstrating the gap in performance as the number of varying masks increases. Note that the large peak in SSIM at the start of 100 plaintext attacks –Fig. I.23b – occurs as the network has not been exposed to enough training examples due to the small dataset size.

[16] https://medium.com/dataseries/convolutional-autoencoder-in-pytorch-on-mnist-dataset-d65145c132ac#63b2; https://machinelearningmastery.com/upsampling-and-transpose-convolution-layers-for-generative-adversarial-networks/

For 100 masks. Due to long training times for the hybrid approach, we use a set of random masks for the 100 mask case. From our experiements we have observed similar decoder performance for learned or random masks. Tab. I.6 compares the image quality metrics for 10 learned and random masks that are varying. As the results and test curves (see Fig. I.24) are very similar, we expect the same for 100 masks.

Table I.6. Image quality (PSNR / SSIM) of trained decoder for varying number of plaintext attacks and number of varying masks. Higher PSNR/SSIM is better.

# attacks ↓	10 (learned)	10 (random)
100	12.5 / 0.21	12.3 / 0.20
1′000	14.2 / 0.32	14.3 / 0.32
10′000	16.2 / 0.43	16.2 / 0.43
100′000	18.0 / 0.53	17.9 / 0.53

Example generated outputs. Fig. 6 shows example generated outputs for 100′000 plaintext attacks. Figs. I.20 to I.22 show the corresponding generated outputs for 10K, 1K, and 100 plaintext attacks respectively.

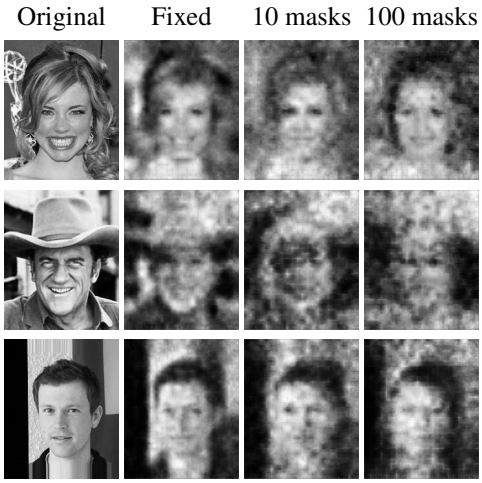


Figure I.20. Example outputs of a decoder that was trained with 10′000 plaintext attacks of embeddings of resolution (24×32).

J. Point spread functions of learned masks

The point spread functions of the learned masks for the experiments in Sections 5.1 to 5.3 can be found in Fig. J.25. All of these masks were initialized with the same seed, hence the similarity in pattern.

The point spread functions of the learned masks for the plaintext generator experiment can be found in Fig. J.26. These were initialized with different seeds. The test curves

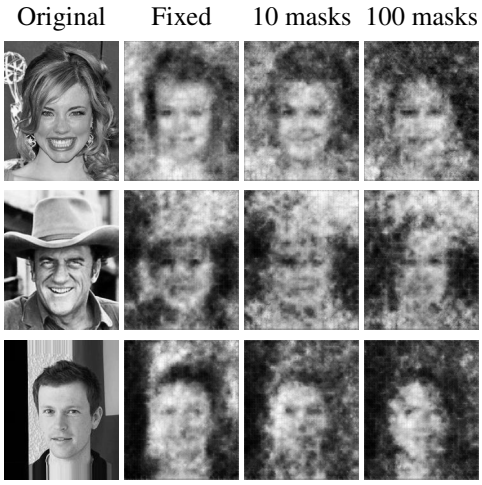


Figure I.21. Example outputs of a decoder that was trained with 1′000 plaintext attacks of embeddings of resolution (24×32).

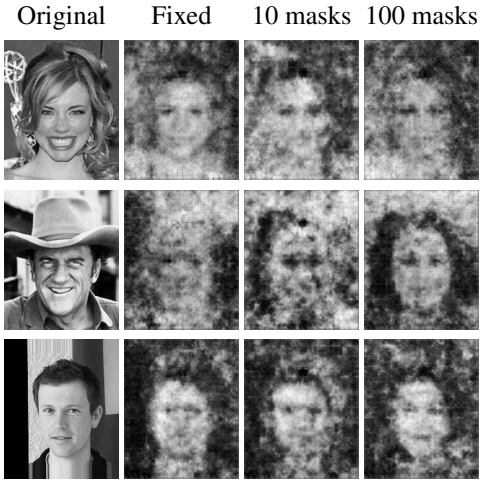
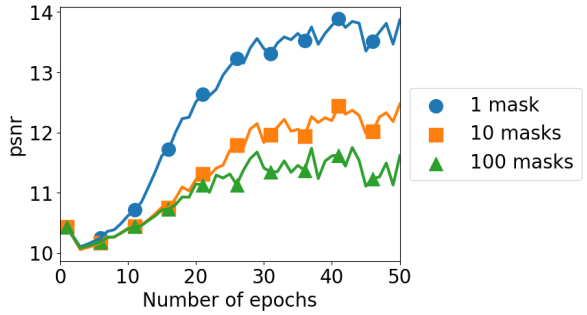
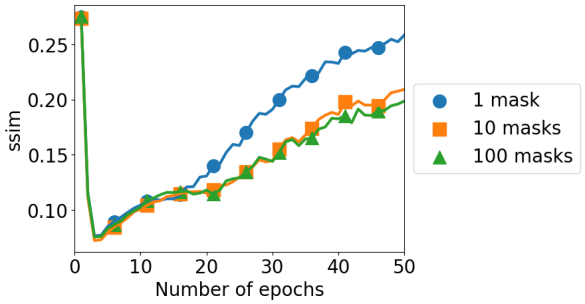


Figure I.22. Example outputs of a decoder that was trained with 100 plaintext attacks of embeddings of resolution (24×32).

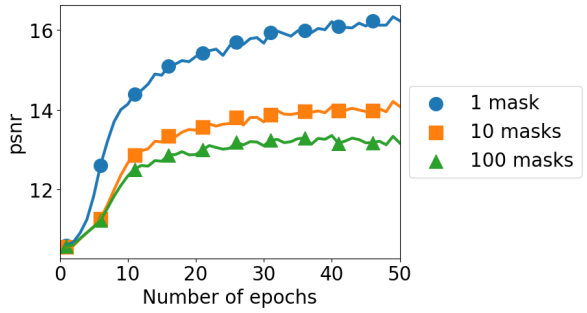
can be found in Fig. J.27, showing the robustness to different initializations.



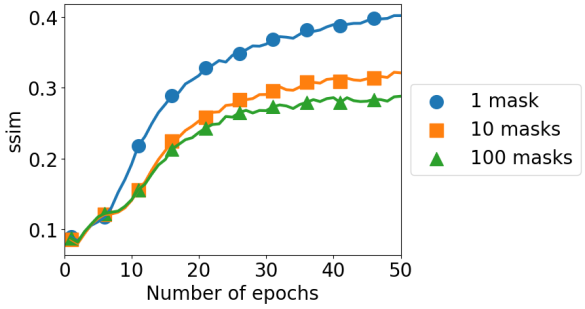
(a) PSNR, 100 plaintext attacks.



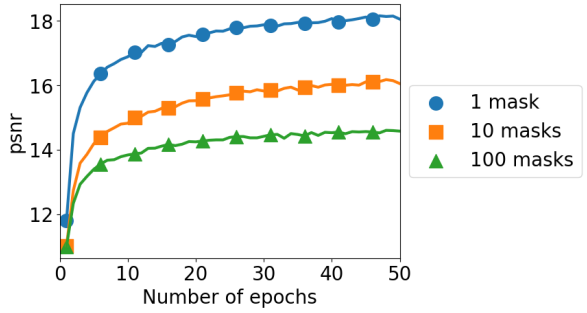
(b) SSIM, 100 plaintext attacks.



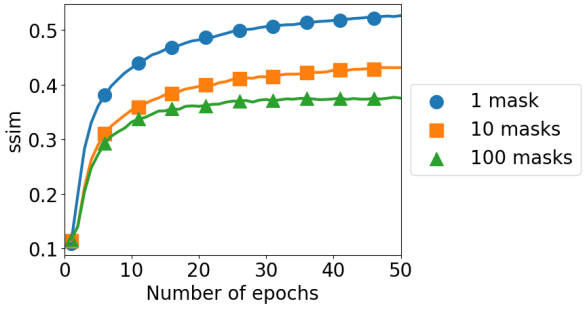
(c) PSNR, 1K plaintext attacks.



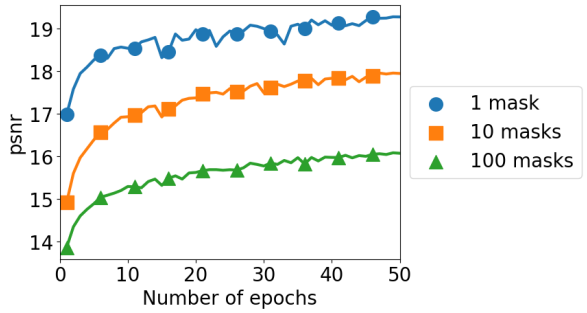
(d) SSIM, 1K plaintext attacks.



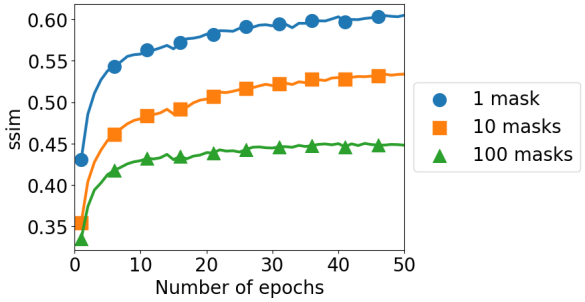
(e) PSNR, 10K plaintext attacks.



(f) SSIM, 10K plaintext attacks.

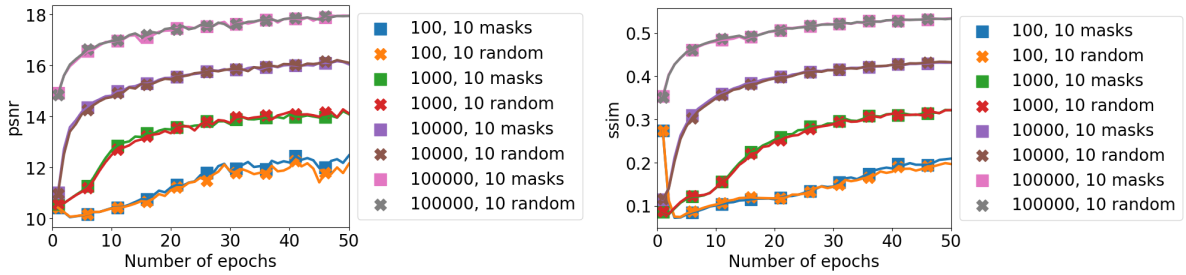


(g) PSNR, 100K plaintext attacks.



(h) SSIM, 100K plaintext attacks.

Figure I.23. PSNR and SSIM curves for training CelebA generator.



(a) PSNR. (b) SSIM.

Figure I.24. Peak signal-to-noise ratio (PSNR) and structural similarity (SSIM) test curves for generators of CelebA; trained with varying number of masks in the imaging system.

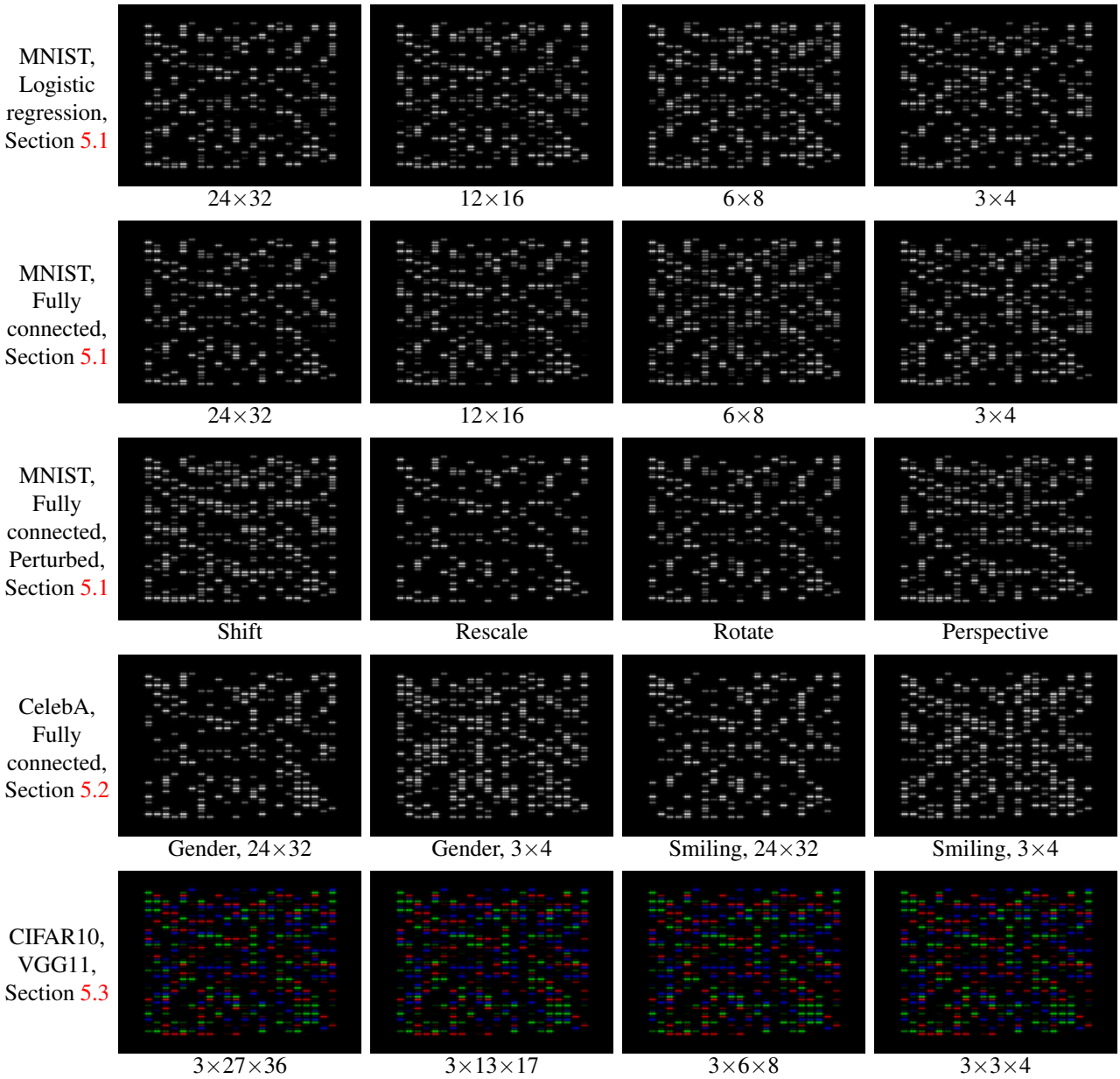
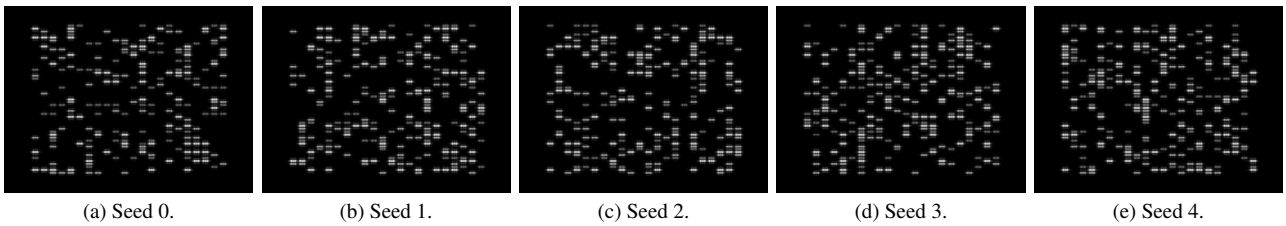


Figure J.25. Point spread functions of learned masks for each experiments in Sections 5.1 to 5.3.



(a) Seed 0. (b) Seed 1. (c) Seed 2. (d) Seed 3. (e) Seed 4.

(f) Seed 5. (g) Seed 6. (h) Seed 7. (i) Seed 8. (j) Seed 9.

Figure J.26. Point spread functions of learned masks for the plaintext generator experiment in Appendix I.2.

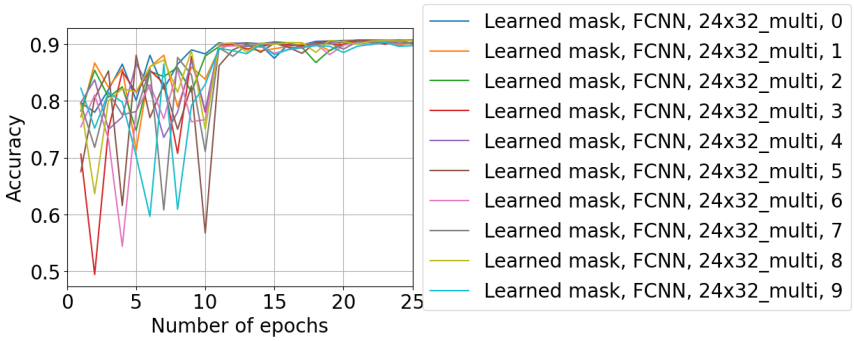


Figure J.27. Test accuracy curves for different random initializations of training *Learned mask* for gender classification with a two-layer fully connected neural network – (24×32) embedding dimension and 800 hidden units. Learning rate is decreased every 10 epochs.