Ninja Codes: Exploring Neural Generation of Discreet Visual Codes

Yuichiro Takeuchi
Sony CSL Kyoto
Japan
yutak@acm.org

ABSTRACT

In this paper we report the results of our early explorations regarding *Ninja Codes*, a new class of visual codes intended to be used in a variety of interactive applications including augmented reality, motion/gesture control, contactless data transfer, robotics, etc. By harnessing the power of adversarial examples, Ninja Codes can be rendered discreet, concealed to human eyes but easily recognizable to detectors based on deep neural networks. The paper will provide a high-level overview of Ninja Codes, and describe an initial, proof-of-concept implementation built on top of existing face detection software. We see this work as a promising step toward a new family of methods by which digital information can be seamlessly encoded into real-world objects and environments.

CCS CONCEPTS

• **Human-centered computing** → **Ubiquitous and mobile computing systems and tools**; • **Computing methodologies** → *Computer vision.*

KEYWORDS

Ninja Codes, visual codes, adversarial examples, neural network

ACM Reference Format:
Yuichiro Takeuchi. 2021. Ninja Codes: Exploring Neural Generation of Discreet Visual Codes. In *CHI Conference on Human Factors in Computing Systems Extended Abstracts (CHI '21 Extended Abstracts), May 8–13, 2021, Yokohama, Japan.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3411763.3451832

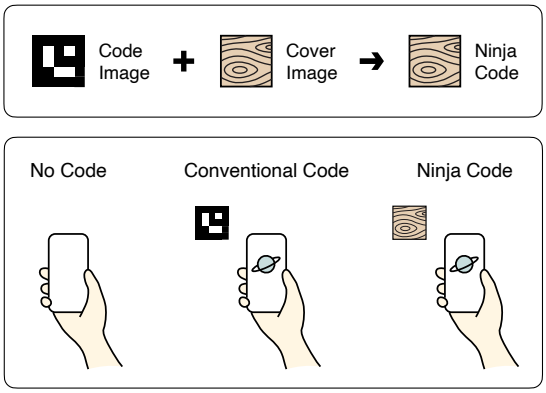

Figure 1: Ninja Codes, visual codes rendered using adversarial example generation that fade into real-world objects and environments. Here, a "cover image" of a wood grain texture is used to generate code that blends into wooden surfaces.

1 INTRODUCTION

2D visual codes—printable, data-embedded graphical patterns that can be detected and decoded using computer vision—have become a common sight throughout the world. The most widely used type of visual codes is QR codes [1], which were initially devised to aid logistics but now enjoy longstanding use in an array of industries; their use has further increased under the ongoing global pandemic, as a convenient method for contactless data transmission in locales such as hotels and restaurants. Visual codes also have a long history of use as fiducial markers that assist localization, i.e., as a means to accurately identify objects' 6DoF positions within environments. This has proved useful for a wide range of applications including augmented reality, gesture/motion control, robotics, etc. Although marker-less localization is considered a more desirable solution, achieving robust performance under non-ideal conditions remains an elusive goal and the need for visual codes persist.

Visual codes tend to have conspicuous appearances, which may be deemed an advantage in some scenarios as they make the codes readily perceptible to humans. However, the inflexible aesthetics of visual codes have also been called out as problematic for some use cases, triggering developments of more discreet and/or aesthetically variable codes [22–24]. In general, such techniques entail a tradeoff between aesthetics and performance: the less "code-like" the codes appear to humans, the more difficult they are for software to locate and decode accurately.

We introduce Ninja Codes, a new class of discreet visual codes whose original appearances are hidden to human eyes using *adversarial example generation* [10]—a technique borne out of neural network research that creates images whose contents will be perceived entirely differently by humans and computer vision software. (For example, an image can be generated that appears to humans as that of a panda bear, but will be recognized as showing an airplane by classification networks such as VGG-19 [20].) Ninja Codes apply this technique to conceal visual codes under arbitrary *cover images* provided by users to suit various usage scenarios (Figure 1). The use of adversarial examples allows codes to be created that naturally blend into real-world objects and environments, while maintaining acceptable detection performance.

CHI '21 Extended Abstracts, May 8–13, 2021, Yokohama, Japan

ACM ISBN 978-1-4503-8095-9/21/05…$15.00
https://doi.org/10.1145/3411763.3451832

This paper will describe our early investigations regarding Ninja Codes. We will give a general overview of its technical principles, and present a proof-of-concept implementation built atop existing open-source face detection software (*BlazeFace* [4]).

2 RELATED WORK

2D visual codes were originally introduced more than 30 years ago in the field of logistics, as a means to allow workers to easily track and identify packages. Over time, their use gradually expanded to include various consumer-facing applications such as advertising and entertainment; this spawned a string of work [22–24] aimed at creating more aesthetically pleasing visual codes that eschew the rigid, utilitarian appearances of common codes such as QR codes. Generally, codes created using such techniques still retain distinctly "code-like" appearances, and their aesthetic range remains limited unless major compromises are made in detection performance. The use of visual codes as fiducial markers goes back to the early 1990s. Codes designed for this purpose [6, 8, 13] tend to be visually simpler with fewer details compared to those used in logistics, etc., due to them prioritizing ease and speed of detection over the amount of encodable data.

In some applications, images are used in lieu of visual codes (e.g., augmented reality apps that use corporate logos as markers). Note that this is a fundamentally different technical setup, that requires images to be preregistered in order for them to be detectable. With visual codes, a single detector can recognize a practically unlimited variety of new codes created on the fly, with no need for software/database updates.

Within the past decade, research on deep convolutional networks have led to considerable advances in computer vision, particularly image recognition. One interesting property of neural networks is their susceptibility to adversarial examples [10], i.e., input images that are strategically designed to induce erroneous responses from targeted networks. Early techniques such as FGSM (Fast Gradient Sign Method) produced brittle images whose adversarial properties vanished even with slight modifications such as rotation or scaling. However, with the introductions of newer techniques such as EOT (Expectation Over Transformation) [2] and attacks that target *object detectors* (e.g., YOLO [17], SSD [15]) in addition to simple classifiers, it has now become possible to create reasonably robust adversarial examples that stay effective even after being subject to substantial perturbations [7, 21].

Ninja Codes use adversarial examples to hide information under arbitrary cover images. The use of adversarial examples to conceal information to human eyes has precedents in steganography research [3, 25]. However, such efforts do not target object detectors, and also operate strictly within the digital realm; in contrast, Ninja Codes are intended to be printed on physical media such as paper, and captured by cameras from various angles and distances under a range of environmental conditions. This leads to different sets of technical requirements.

3 NINJA CODES

Before Ninja Codes can be created, as prerequisites we first need a scheme of 2D visual codes (*code scheme*), and also a corresponding neural network trained to detect codes that adhere to said scheme (*detector*). The detector should take an image (which may or may not contain visual codes) as input, and output a tensor containing both the locations of detected codes (if any) and data encoded in each of them.

One way to satisfy these requirements is to start with an existing code design (e.g., QR codes), prepare a dataset of annotated images (in which the codes appear in various sizes/positions/orientations under a range of environmental conditions), and use the dataset to train an object detector network. Or alternatively, a custom code scheme may be newly developed. In any case, the performance of the code scheme / detector network combination will serve as the upper limit of the detection performance of Ninja Codes.

Once we have a functioning combination of a code scheme and a detector, we can move on to actually creating Ninja Codes. The user supplies two images: 1) a *code image* (x_{code}), i.e., an image of the visual code (adhering to the chosen code scheme) that encodes user-defined binary data, and 2) a *cover image* (x_{cover}) that specifies what the resulting Ninja Code should look like to the human eye (e.g., a wood grain texture may be used if we wish to create a Ninja Code that naturally blends into surfaces of wooden furniture.) An iterative optimization process then generates an *adversarial example* (x), that appears as close as possible to x_{cover} (to humans) while triggering responses from the detector similar to those triggered by x_{code}. (In other words, the detector network is unable to distinguish between x and x_{code}.) Using Lagrangian relaxation [5], this optimization problem can be formalized as follows:

$$\underset{x}{\operatorname{argmin}} \ \mathbb{E}_{t\sim T} \sum_{k=0}^{n} \lambda_k \mathcal{L}_k \tag{1}$$

Here, T is a distribution of differentiable image transformation functions t, chosen to simulate both the range of angles/distances from which Ninja Codes will be captured in real-life scenarios, and possible variations in environmental lighting, camera/lens, printing material, etc. (Generated Ninja Codes will typically be undetectable under conditions not anticipated in T.) Scalars $\lambda_0, \lambda_1, \ldots, \lambda_n$ are empirically chosen weights for loss terms $\mathcal{L}_0, \mathcal{L}_1, \ldots, \mathcal{L}_n$, which are defined as follows:

- $\mathcal{L}_0$: The distance between adversarial example x and cover image x_{cover}, both transformed using t
- $\mathcal{L}_1$: The distance between the respective responses of the detector to adversarial example x and code image x_{code}, both transformed using t
- $\mathcal{L}_2, \mathcal{L}_3, \ldots, \mathcal{L}_n$: Other losses necessitated by the specifics of the code scheme and/or the application domain, e.g., non-printability score [19]

A judicious choice of distribution T that aptly reflects intended usage scenarios should make x robust to perturbations, resulting in Ninja Codes that can reliably be detected even when printed on a multitude of materials and captured under varying conditions. Although adversarial examples are known to transfer reasonably well [16], for best performance the same detector network used to generate Ninja Codes should also be used to recognize them. Note that the above is a high-level description of Ninja Codes and there exists considerable latitude with regards to implementation details,

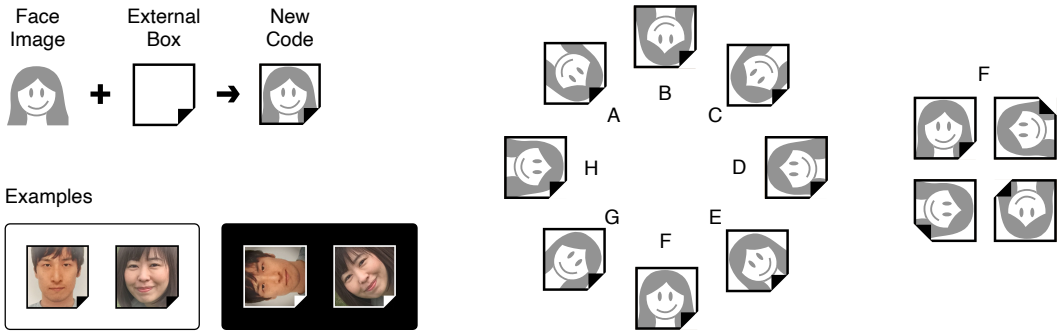

Figure 2: Code scheme for our proof-of-concept implementation. The relative orientation of the face to the solid-colored box is used to encode 3 bits of information.

such as code scheme, detector architecture, optimization algorithm, loss term definitions, weight values, etc.

4 PROOF-OF-CONCEPT IMPLEMENTATION

As a proof-of-concept of Ninja Codes, we present a simple implementation built using BlazeFace [4], a multi-platform, open-source face detection software based on the MobileNet architecture [11, 18]. Here we use its iOS implementation, whose frame rate is reported to exceed 1000fps on an off-the-shelf smartphone (Apple iPhone XS). The BlazeFace network takes 256×256px images as input, and outputs tensors containing information about the locations of detected faces, each accompanied by a confidence score and also a set of six keypoints corresponding to positions of facial features (eyes, ears, nose, mouth).

4.1 Code Scheme

Figure 2 illustrates the code scheme we use in our proof-of-concept implementation. Instead of a black-and-white matrix design commonly seen in visual codes, here we use a specialized scheme where each code takes the form of a solid-colored box containing a single image of a person's face. Information is encoded using the face's orientation in relation to the external box. This simple code scheme allows us to quickly implement Ninja Codes without going through the process of preparing an annotated dataset and training a detector network ourselves.

The code scheme comes with obvious downsides, the largest of which is that it can only encode small bits of information (3 bits in our implementation). In theory, the size of encodable data can be increased to moderate extents by detecting more minute differences in face orientations, taking into account face size/position, identifying facial features, including multiple faces inside the box, etc. Such enhancements are outside the scope of this paper, however, as our goal here is to demonstrate a quick-and-dirty implementation of the overall concept. Another downside is the need for the external box, which makes the codes less discreet compared to *true* Ninja Codes. (The box is used as an anchor to determine the relative orientation of the contained face image, and also to differentiate the codes from actual human faces which may exist in the scene.) The box is drawn in either solid white or black, and is marked in one corner to make its orientation discernible; for each face detected by BlazeFace, a

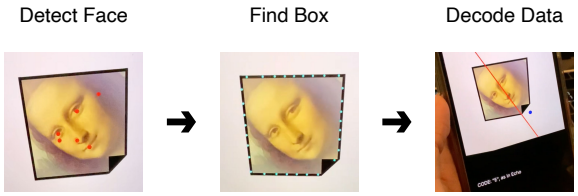

Figure 3: For each detected face, a CPU-based process checks whether it is surrounded by an external, solid-colored box. If a box is found, the relative orientation of the face to the box is evaluated to extract encoded information. Ninja Codes are detected/decoded in the exact same manner.

lightweight CPU-based process checks for the existence of the box and reads out encoded data if one has been found (Figure 3).

Creating a new code in this scheme is a straightforward process of rotating/cropping an arbitrary image containing a person's face, and embedding it within a solid-colored box. BlazeFace can detect faces in both color and grayscale images, and in our experiments worked reasonably well with realistically-drawn illustrations and cartoons in addition to photographs. One caveat is that BlazeFace works best for faces that are more or less upright; in order to detect non-upright (e.g., upside-down) faces, our detection software passes a single video frame to BlazeFace 4 consecutive times, each time rotating the entire frame 90 degrees.

The face-based code scheme was devised as a stopgap solution to assist us in our proof-of-concept implementation. Needless to say, for best performance preparing a custom code scheme along with a specifically-trained detector network would be desirable, whose development comprises a central component of our future work. However, there may exist scenarios (e.g., quick prototyping) where the ability to effectively piggyback on proven face detection software with highly-optimized performance becomes advantageous; as face detection is an in-demand, fast developing field of technology, we can expect continuous advances to be made that can be further exploited by this code scheme.

4.2 Adversarial Example Generation

Codes created through the above process are not yet discreet; they are turned inconspicuous by replacing the contained face images

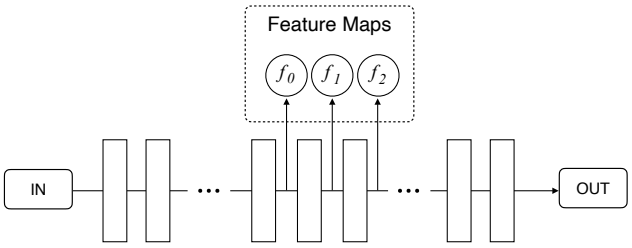

Figure 4: Intermediate feature maps of a detector network. In generating adversarial examples, instead of minimizing the distance between the detector's final outputs, we attempt to minimize the sum of distances for a select set of intermediate outputs.

with corresponding *adversarial examples*. Our method for generating adversarial examples mostly mirrors conventional techniques, but entails slight modifications inspired by work on neural style transfer [9, 12].

A key intuition of neural style transfer is that responses of intermediate layers of neural networks (*feature maps*, Figure 4) hold interesting information that can be put to use in image generation tasks. For example, given two images—a photograph of a person's face and Van Gogh's *Starry Night*—a new image showing the person's face redrawn in Van Gogh's style can be synthesized, through computations involving the intermediate responses of image classifier networks such as VGG-19. In other words, the style and content of arbitrary images can be isolated by giving them as input to neural networks and studying their intermediate feature maps. Applying this intuition to our problem at hand, we generate adversarial examples by defining $\mathcal{L}_1$ as follows:

$$\mathcal{L}_1 = \sum_{l \in L} \sum_{i} [f_l^i(t(x)) - f_l^i(t(x_{code}))]^2 \tag{2}$$

Here, x_{code} is the code image, which in our case is an image containing a human face (cropped/resized to 256×256px) and rotated so that the face sits upright and is positioned roughly in the center of the image. f_l^i refers to the ith element of the detector's (BlazeFace's) intermediate output (feature map) at layer l. L denotes the set of intermediate layers whose outputs will be taken into account in calculating $\mathcal{L}_1$; in our implementation, we use the outputs from the last five *BlazeBlocks* (i.e., modules containing multiple convolutional layers and pooling/padding operations) of the BlazeFace network. Intuitively, our choice to define $\mathcal{L}_1$ in this way—as opposed to a more standard definition that takes the distance between the network's final outputs—can be understood as attempting to create more robust adversarial examples at the risk of introducing some level of *face-like* features in the output. (Although we have yet to conduct formal evaluations, our informal observations so far strongly support our case for higher robustness.) Redefinitions of set L can be used as a fine-tuning mechanism; including lower, i.e., earlier layers of the network in L will likely lead to better detection performance, albeit also making the final image look increasingly *face-like* to human eyes.

Regarding the other loss terms, $\mathcal{L}_0$ is defined straightforwardly as the mean squared error between $t(x)$ and $t(x_{cover})$, i.e., images x and x_{cover} both transformed using t. For $\mathcal{L}_2$, we introduce a *color divergence score* derived by first calculating $x - x_{cover}$ (*perturbation image*), and then adding up the RGB variances of all pixels within this image. (In the equation below, x^i and x_{cover}^i refers to the ith pixel of images x and x_{cover}, respectively.) Note that we do not apply t to images when calculating the color divergence score; this simplifies computation, and in our experience does not result in any visible drop in visual quality or detection performance.

$$\mathcal{L}_2 = \sum_{i} \mathrm{Var}(x^i - x_{cover}^i) \tag{3}$$

In effect, by introducing the above color divergence score, we are attempting to minimize the introduction of unwanted colors in the final image by trying to keep perturbations close to grayscale values; this helps counter patches of skin-like colors emerging within x. Although Ninja Codes are primarily intended to be printed on paper and other physical media, to reduce computational cost we opted not to use the non-printability score described in [19].

For T we use a randomized collection of differentiable transformation functions, i.e., rotation, scaling, perspective transformation, noise addition, and contrast/brightness adjustment. A broader distribution of T will expand the range of conditions (e.g., angle, distance, lighting) under which Ninja Codes can be detected, at the risk of introducing more face-like features. Weights λ_0, λ_1, λ_2 are adjusted independently for each image; as a rule of thumb, we begin with $\lambda_0 = 1000$, $\lambda_1 = 10$, $\lambda_2 = 0.1$, and iteratively increase/decrease λ_0 until we achieve satisfactory results. Optimization is performed via stochastic gradient descent using Adam [14] on PyTorch (learning rate starts at 0.1, and is set to progressively decrease). Generating a single adversarial example takes approximately 18 minutes on our 2020 Macbook Pro (Intel Core i9, CPU only). Once the algorithm converges, the final Ninja Code can be created by cropping/rotating the output image in accordance to the 3-bit data we wish to encode, and enclosing it within a solid-colored external box.

Figure 5 shows several examples of generated Ninja Codes. We found the codes to be reliably recognizable and decodable using a BlazeFace-based detection software running on iPhone 12 Pro. The codes successfully mimic the general visual characteristics of the user-selected cover images. However, upon close inspection some face-like features introduced during the optimization process are visible. Such features can be quite prominent for cover images with relatively uniform textures.

5 DISCUSSION

Our proof-of-concept implementation suffers from several obvious limitations; the most detrimental of which is the minuscule size of encodable data. We plan to conduct a fully-fledged implementation with increased data capacity as part of future work, but there may exist select usage scenarios where the current capacity is already acceptable, e.g., some gesture control applications where one needs to only detect and differentiate between a handful of codes.

Exactly to what extent we can increase the data capacity through future implementations is unknown, but we believe it reasonable to assume that a capacity of several bytes (enough to encode short

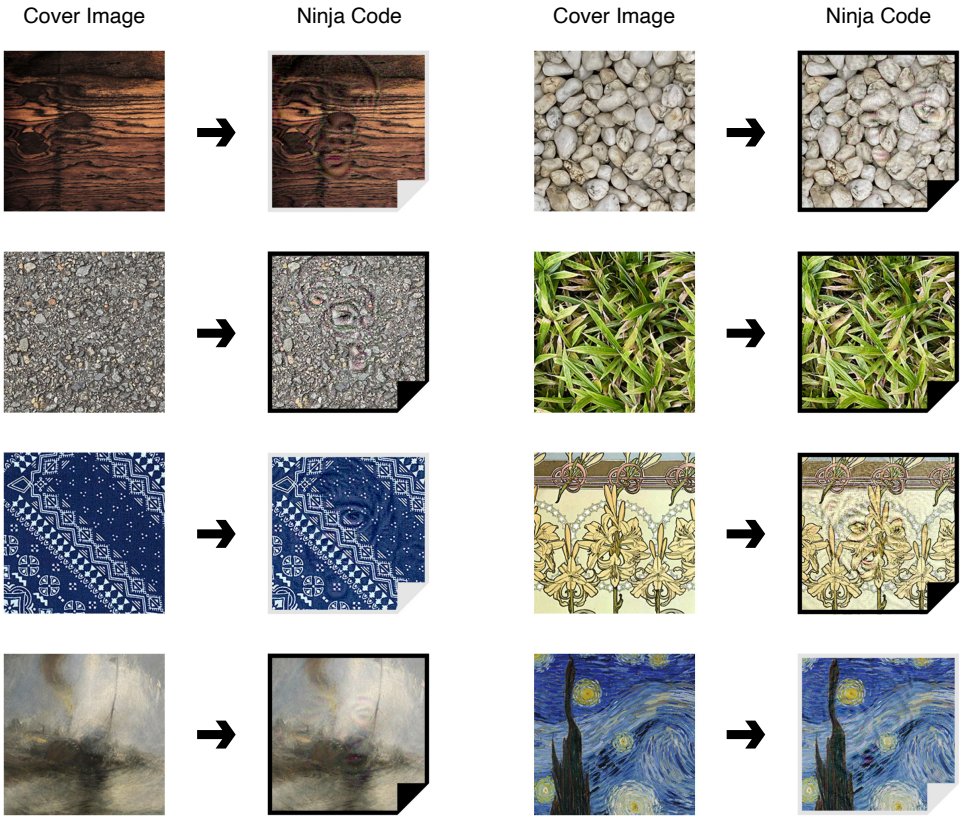

Figure 5: Examples of generated Ninja Codes. Photos of the author were used to create code images (not shown here).

strings but not complete URLs) is easily within reach, which should broaden the range of potential applications. Although there may be little reason to choose Ninja Codes over existing codes in laboratory settings (or other settings where aesthetics are of little concern), the use of interactive technologies have now expanded to include many situations where eschewing distinctly "code-like" appearances will be desirable—for example, artists incorporating motion tracking into their performances or installations may desire codes that do not clash with their works' aesthetic styles, and researchers engaging in ethnographic HCI research may sometimes wish to conceal obvious signs of technological intervention.

Presently, visual codes enjoy a broader range of use cases in East Asia (particularly China) compared to the rest of the world. As such, the various ways in which codes are used in the region may offer us insights about potential future uses of Ninja Codes. For example, in Chinese cities it is not uncommon to see billboard-sized QR codes pinned to building walls, effectively overlaying streetscapes with digital information. Ninja Codes may similarly be deployed at large scales, to provide additional information and/or entertainment to passersby but in a more discreet manner.

An interesting fact regarding most existing visual codes is that they are detected using CPU-based, handcrafted algorithms. Since Ninja Codes are detected using neural network inference on GPUs, with sufficient optimization we should be able to expect superior frame rates, particularly on mobile and embedded devices that lack powerful CPUs.

Prior work have shown that the notion of adversarial examples can be extended to 3D [2], i.e., we can generate objects (and fabricate them using 3D printers, CNC mills, etc.) that will be recognized as arbitrary entities by neural networks. Applying such techniques to Ninja Codes, we may be able to create a varied range of sculptures, furniture, artificial flowers, and other environmental elements that seem entirely ordinary to human eyes but are encoded with hidden information that can be deciphered by neural networks.

6 CONCLUSION

This paper introduced Ninja Codes, a new class of visual codes that can be made inconspicuous through the use of adversarial examples. We described the general concept, and presented a proof-of-concept implementation built atop existing face detection software. Future work will center around preparing a custom combination of code scheme and detector network; we also plan to explore the concept's applicability to 3D. Though advances in deep learning has brought sizable leaps in computer vision, marker-less localization is still an unsolved problem and visual codes remain an important technology in a wide array of domains. We believe Ninja Codes can help make visual codes a more pervasive part of our everyday lives, enabling numerous novel applications but in ways compatible with people's aesthetic sensibilities.

REFERENCES

[1] QR codes. https://www.denso-wave.com/en/technology/vol1.html

[2] Athalye, A., Engstrom, L., Ilyas, A., Kwok, K. Synthesizing Robust Adversarial Examples. Proc. of ICML 2018, pp. 284–293.

[3] Baluja, S. Hiding Images in Plain Sight: Deep Steganography. In Proc. of NIPS 2017, pp. 2066–2076.

[4] Bazarevsky, V., Kartynnik, Y., Vakunov, A., Raveendran, K., Grundmann, K. BlazeFace: Sub-millisecond Neural Face Detection on Mobile GPUs. arXiv:1907.05047, 2019.

[5] Carlini, N., Wagner, D. Defensive Distillation is Not Robust to Adversarial Examples. arXiv:1607.04311, 2016.

[6] DeGol, J., Bretl, T., Hoiem, D. ChromaTag: A Colored Marker and Fast Detection Algorithm. In Proc. of ICCV 2017, pp. 1472–1481.

[7] Eykholt, K., Evtimov I., Fernandes E., Li Bo., Rahmati A., Xiao C., Prakash A., Kohno T., Song, D. Robust Physical-World Attacks on Deep Learning Visual Classification. In Proc. of CVPR 2018, pp. 1625–1634.

[8] Fiala, M. ARTag, A Fiducial Marker System Using Digital Techniques. In Proc. of CVPR 2005, pp. 590–596.

[9] Gatys, L., Ecker, A., Betge, M. Image Style Transfer Using Convolutional Neural Networks. In Proc. of CVPR 2016, pp. 2414–2423.

[10] Goodfellow, I. J., Shlens, J., Szegedy, C.. Explaining and Harnessing Adversarial Examples. arXiv:1412.6572, 2014.

[11] Howard, A. G., Zhu, M., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., Adam, H. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. arXiv:1704.04861, 2017.

[12] Johnson, J., Alahi, A., Fei-Fei, L. Perceptual Losses for Real-Time Style Transfer and Super-Resolution. In Proc. of ECCV 2016, pp. 694–711.

[13] Jorda, S., Geiger, G., Alonso, M., Kaltenbrunner, M. The reacTable: Exploring the Synergy between Live Music Performance and Tabletop Tangible Interfaces. In Proc. of TEI 2007, pp. 139–146

[14] Kingma, D., Ba, J. Adam: A Method for Stochastic Optimization. In Proc. of ICLR 2015, pp. 1–15.

[15] Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C. Y., Berg, A. C. SSD: Single Shot MultiBox Detector. In Proc. of ECCV 2016, pp. 21–37

[16] Papernot, N., McDaniel, P., Goodfellow, I. Transferability in Machine Learning: From Phenomena to Black-Box Attacks Using Adversarial Samples. arXiv:1605.07277, 2016.

[17] Redmon, J., Divvala, S., Girshick, R., Farhadi, A. You Only Look Once: Unified, Real-Time Object Detection. In Proc. of CVPR 2016. pp. 779–788.

[18] Sander, M., Howard, A., Zhu, M., Zhmoginov, A., Chen, L. C. MobileNetV2: Inverted Residuals and Linear Bottlenecks. arXiv:1801.04381, 2018.

[19] Sharif, M., Bhagavatula, S., Bauer, L., Reiter, M. K.. Accessorize to a Crime: Real and Stealthy Attacks on State-of-the-Art Face Recognition. In Proc. of ACM CCS 2016, pp. 1528–1540.

[20] Simonyan, K., Zisserman, A. Very Deep Convolutional Networks for Large-Scale Image Recognition. In Proc. of ICLR 2015, pp. 1–14.

[21] Thys, S., Ranst, W. V., Goedeme, T. Fooling Automated Surveillance Cameras: Adversarial Patches to Attach Person Detection. arXiv:1904.08653, 2019.

[22] Xu, M, Li, Q., Niu, J., Liu, X., Xu, W., Lv, P., Zhou, B. ART-UP: A Novel Method for Generating Scanning-robust Aesthetic QR codes. arXiv:1803.02280, 2018.

[23] Xu, M., Su, H., Li, Y., Li, X., Liao, J., Niu, J., Lv, P., Zhou, B. Stylized Aesthetic QR Code. IEEE Trans. Multimedia 21(8), pp. 1960–1970. 2019.

[24] Yang, Z., Bao, Y., Luo, C., Zhao, X., Zhu, S., Peng, C., Liu, Y., Wang, X. ARTcode: Preserve Art and Code In Any Image. Proc. UbiComp 2016, pp. 904–915.

[25] Zhu, J., Kaplan, R., Johnson, J., Fei-Fei, L. HiDDeN: Hiding Data With Deep Networks. In Proc. of ECCV 2018, pp. 657–672.