A Tour of Convolutional Networks Guided by Linear Interpreters – Appendix –

Pablo Navarrete Michelini, Hanwen Liu, Yunhua Lu, Xingqun Jiang
BOE Technology Co., Ltd.
{pnavarre, liuhanwen, luyunhua, jiangxingqun}@boe.com.cn

Abstract

We provide the following additional information:

- *Classification:*
  - *Explanation of Forward/Back–Projections;*
  - *Residual contributions for more architectures;*
  - *Contribution histograms for more networks;*
  - *Pixel votes for more images;*
  - *What happens after an adversarial attack?*
- *Super–Resolution (SR):*
  - *Projective/receptive filters;*
  - *More eigen–inputs/outputs.*
- *Image–to–Image Translation (I2I):*
  - *Projective/receptive filters;*
  - *Eigen–inputs/outputs.*
- *Demonstrations:*
  - *Video material;*
  - *Interactive material.*

A. Classification

Explanation of Forward/Back–Projections:

In our analysis of linear interpreters for classifiers we use a theorem that is essential to understand how do we decompose the contributions of the network to the output scores. Roughly speaking, the theorem says that:

> *In a sequential network there are explicit expressions for F and r in the linear interpreter $y = Fx + r$. The filter matrix F is given by the forward–projection of the input to the output score. And r is given by the sum of all forward–projected masked–biases from each layer to the output score.*

By *projection* we mean the progressive application of the linear transformation for each layer. **Forward projection** means that we apply the linear transformations of a given layer, and then the transformation of the next layer, and so forth. **Backward projection** means that we apply the transposed linear transformation of a given layer, and then the same in the previous layer, and so forth. Finally, **masked–biases** are the bias parameters of the network (scalars) multiplied by activation masks (images of ones and zeros for ReLU). Then, the masked–biases, denoted by $\hat{b}$, are images in the network's feature domain at the layer, that can be forward or back projected through the network.

The proof of the theorem is straightforward using inductive arguments. So here we prefer to follow a more didactic approach. Namely, we will unfold the formula for the linear interpreter and see how the expressions for filter matrix and residual decomposition appear.

We start with a sequential convolutional network model:

$$y_n = W_n x_{n-1} + b_n \qquad \text{and} \qquad x_n = h\left(y_n\right) \ , \tag{1}$$

with parameters b_n (biases) and sparse matrices W_n (convolutions, including strided and transposed).

Let $\hat{W}_n = A_n W_n$ and $\hat{b}_n = A_n b_n + c_n$. Where A_n, c_n are the parameters of the linear interpreter for $h(y_n)$. Then we have:

$$x_n = h\left(W_n h\left(W_{n-1}x_{n-2} + b_{n-1}\right) + b_n\right) \tag{2}$$

$$= \hat{W}_n \left(\hat{W}_{n-1}x_{n-2} + \underbrace{A_{n-1}b_{n-1} + c_{n-1}}_{\hat{b}_{n-1}} \right) + \underbrace{A_n b_n + c_n}_{\hat{b}_n} \tag{3}$$

$$= \hat{W}_n \hat{W}_{n-1} \underbrace{x_{n-2}}_{\hat{W}_{n-2}x_{n-3}+\hat{b}_{n-2}} + \hat{W}_n \hat{b}_{n-1} + \hat{b}_n \tag{4}$$

$$= \prod_{k=1}^{n} \hat{W}_k x_0 + \prod_{k=2}^{n} \hat{W}_k \hat{b}_1 + \prod_{k=3}^{n} \hat{W}_k \hat{b}_2 + \cdots + \hat{b}_n \ . \tag{5}$$

Now we can define:

$$Q_n = I\ , \quad Q_i = \prod_{k=i+1}^{n} \hat{W}_k, \quad \text{for } i = 1, \dots, n\ . \tag{6}$$

and we get:

$$\boxed{x_n = \underbrace{Q_0}_{F}\, x_0 + \underbrace{Q_1\hat{b}_1 + Q_2\hat{b}_2 + \cdots + \hat{b}_n}_{r}}\ . \tag{7}$$

The filter matrix and residual at layer n are then given by:

$$F = \prod_{k=1}^{n} \hat{W}_k\ , \quad \text{and} \quad r = \sum_{i=1}^{n} Q_i \hat{b}_i\ . \tag{8}$$

This expression follows the so–called *conservation property* of LRP[1] because the final score (the output of the network) is written as a sum of layer–wise contributions. Nevertheless, the nature of these contributions here has a different meaning, not as relevances but as forward–projected masked–biases.

Matrices Q represent forward–projections, since they progressively apply linear transformations towards the output. Similarly, we can define:

$$P_0 = I\ , \quad P_i = \prod_{k=1}^{i} \hat{W}_k^T, \quad \text{for } i = 1, \dots, n\ . \tag{9}$$

Here, **matrices P represent back–projections**, since they progressively apply transposed linear transformations towards the input.

We can use this definition to give an explicit expression for the **Pixel Discussion** (PD) images displayed in the main text. This is

$$\boxed{\text{PD} \propto P_0 F^T(x_0) + P_1\hat{b}_1 + P_2\hat{b}_2 + \cdots + P_n\hat{b}_n}\ , \tag{10}$$

and PD is normalized so that the sum of all of its pixels gives us the output score. Then, each pixel value in PD gives a pixel–wise contribution to the final score.

Residual contributions for more architectures:

In Table 1 we show the average contribution of residual to classification scores for a more complete list of architectures, including standard deviation values. In VGG and SqueezeNet we observe that the residual contribution increases for larger networks (with better benchmarks) but this pattern does not repeat for other architectures. Standard deviations are smaller for larger contributions of the residual, indicating that these architectures are consistently using the residual to improve their classification scores.

Contribution histograms for more networks:

AlexNet 78.5% ±15.8	SqueezeNet 1.0 80.7% ±11.5	VGG–11 82.2% ±14.1
ResNet–18 80.5% ±13.9	SqueezeNet 1.1 84.3% ±11.0	VGG–13 82.11% ±13.3
ResNet–34 83.7% ±12.9	DenseNet–121 94.6% ±4.5	VGG–16 84.2% ±12.0
ResNet–50 82.0% ±16.4	DenseNet–161 95.0% ±4.0	VGG–19 85.5% ±10.9
ResNet–101 80.2% ±13.6	DenseNet–169 94.4% ±4.4	VGG–11–BN 84.5% ±12.6
ResNet–152 81.1% ±16.9	DenseNet–201 94.0% ±4.8	VGG–13–BN 85.1% ±12.5
	Inception v3 91.6% ±8.0	VGG–16–BN 86.2% ±12.9
		VGG–19–BN 85.8% ±13.6

Table 1: Average contributions of residuals to classification scores for 100 validation images from ImageNet–1k[6]. Numbers below percentage represent standard deviation. DenseNet and Inception architectures show highest contributions of the residual, with smaller standard deviation.

In Figure 1 we show histograms of the layer–wise contributions to top–1 scores for a series of VGG network architectures. We use pre–trained models trained with and without batch–normalization[1]. We observe in most cases that the contribution of the input, $F(x_0)x_0$, does not account for the largest part of the final score. So it is necessary to use the layer–wise decomposition of the residual to really see where do the contributions come from.

When trained without batch–normalization, we consistently see two major contributions. One in early layers of the network (before the first pooling layer). And the second major contribution comes from much deeper in the network, just before the fully connected layers. This pattern clearly changes in networks trained with batch–normalization. In this case the contributions move inside the network, with major contributions just before fully connected layers.

Pixel votes for more images:

In Figure 2 we show more examples of pixel discussions and pixel votes. Here, we observe more evidence that pixel dicussions (PDs) are not conclusive about the network's decision. Sometimes, pixels seem to discuss strongly on an object (e.g. wolf, pickup car, taxi, etc.) but in other cases

[1]Classifier models downloaded from https://pytorch.org/docs/stable/torchvision/models.html

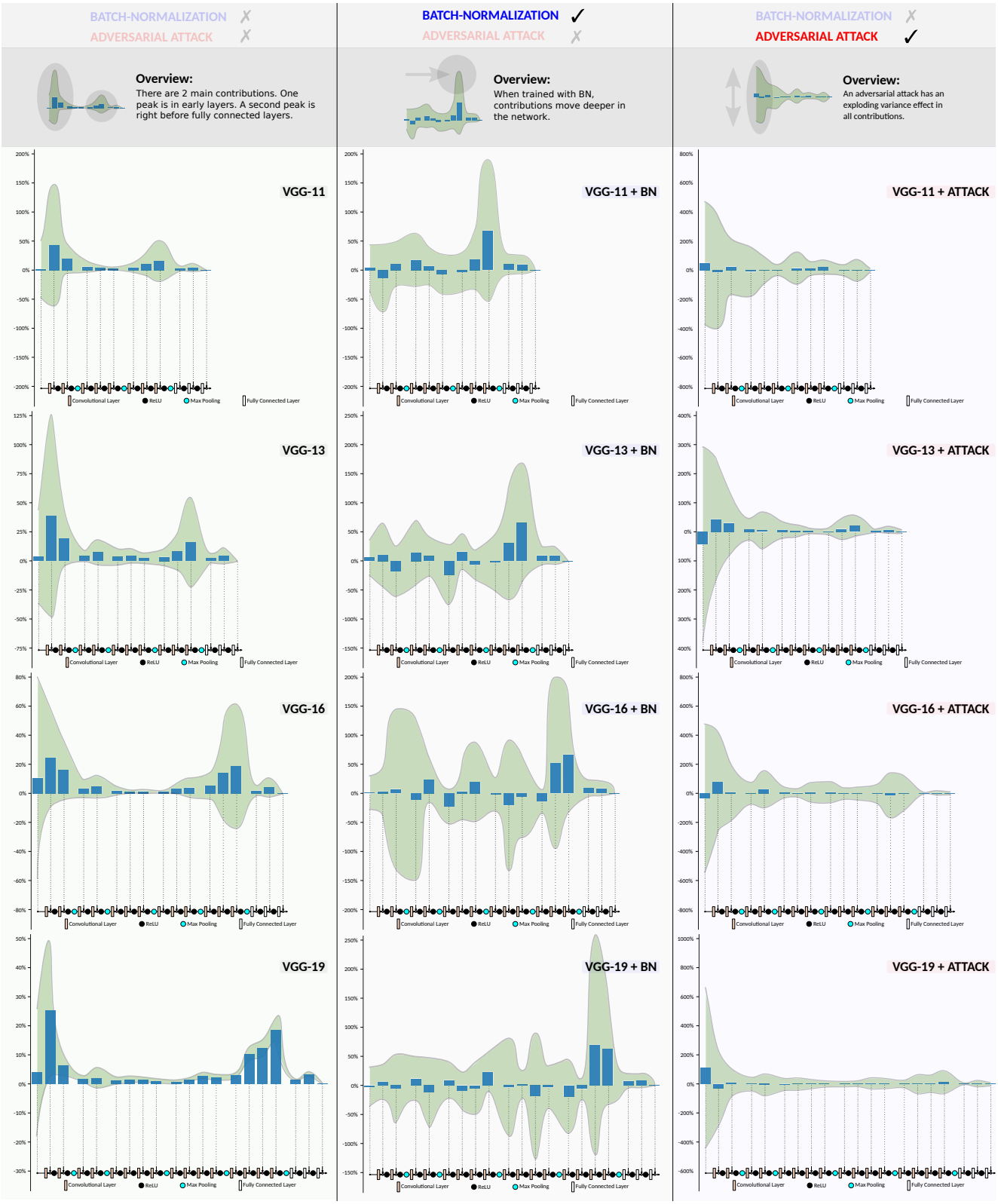

Figure 1: Layer–wise contributions to Top–1 scores for pre–trained VGG classifiers, averaged over 100 images from ImageNet–1k. Standard deviation shown as shaded area. The first column shows models trained with original images and without batch–normalization. The second column uses original images and models with batch–normalization. The third column considers the same group of 100 images with adversarial attack added by using FGSM[2].

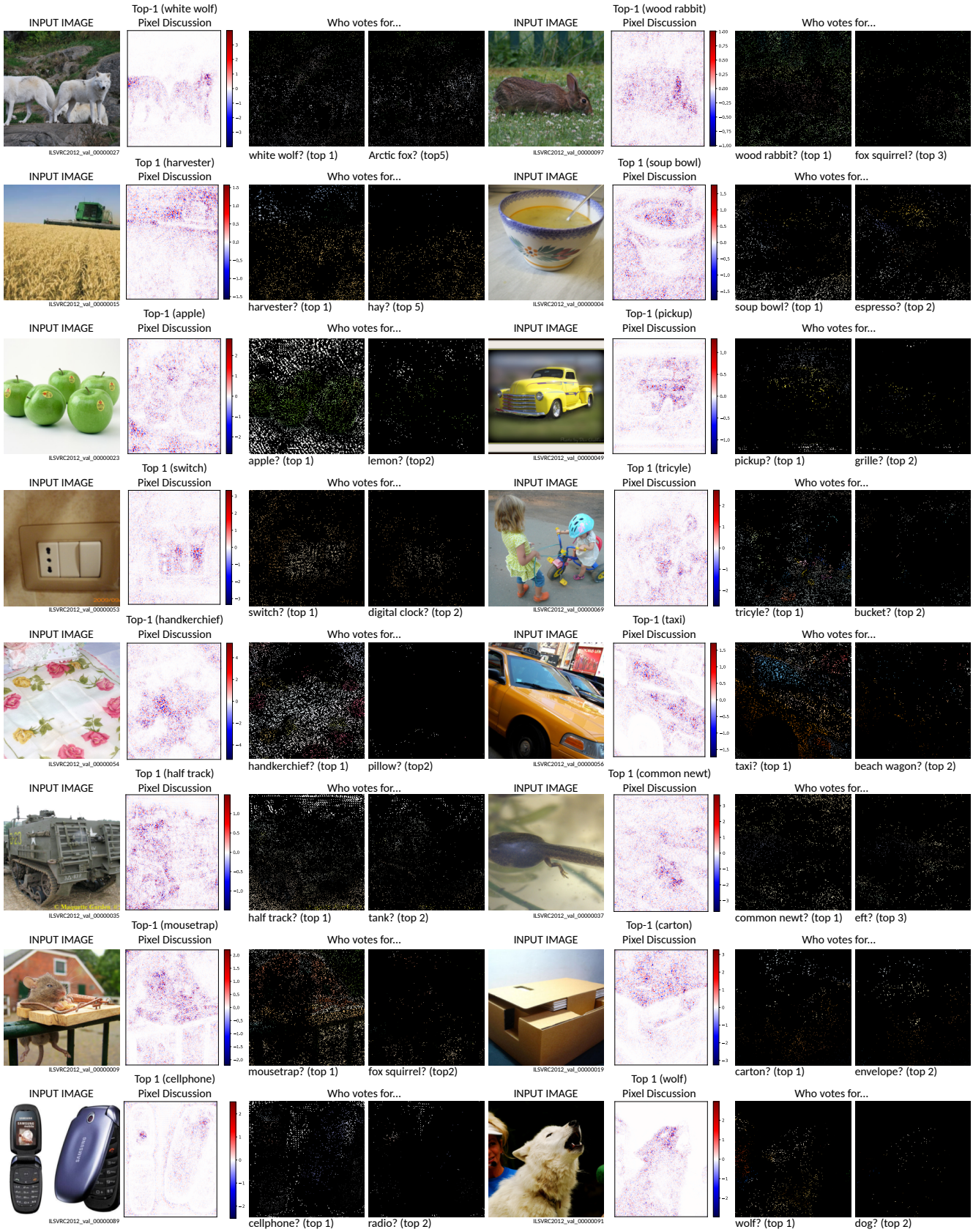

Figure 2: Pixel–discussions are back–projections of output scores to input domain that show pixel–wise contributions to the scores. By comparing contributions among all scores, we make pixels vote independently and find that they focus on objects.

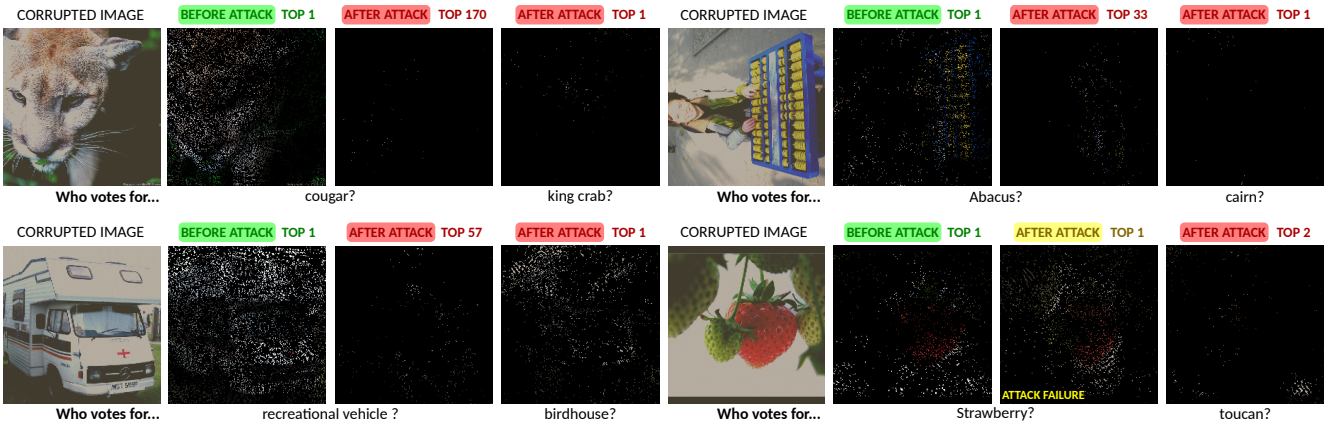

Figure 3: Effect of an adversarial attack using FGSM[2] on the votes of pixels. When an attack succeeds the pixels clearly stop to vote for the right label. In most cases the pixels do not seem to vote much for the new top–1 label, suggesting that the attack is spreading the opinion of pixels throughout all the $1,000$ classes.

the discussion takes place outside the main object (e.g. harvester, soup bowl). After we compare the discussions over all labels we can see what is the overall pixel–wise outcome of the discussion. These so–called *pixel votes* focus on the main objects and show clear preferences for the top score in the output of the network. We observe how "harvest" and "hay" labels have pixels focused on areas with hay; an "espresso" label have pixels looking at the liquid in a bowl; a "digital clock" label have pixels on a square–shape plug that looks like a digital clock; etc.

What happens after an adversarial attack?

We have observed clear patterns in the contribution histograms and pixel votes that show the layers where networks make decisions and the pixel–wise preferences for each label. To explore these patterns further, now we consider the effect of an adversarial attack on the network. Namely, we consider a *Fast Gradient Sign Method*[2] (FGSM)[2] that introduces noise in input images, making them look brighter but keeping the content visible to human eyes. In Figure 3 we observe how the attack changes the decision of the network without displaying visible content of the new top labels in the corrupted images. Here, we observe a strong change in the pattern of pixel votes. In one case (strawberry) where the attack fails, we still see the pixels voting more for top–1 label. When the attack succeeds, pixels stop to vote for the right label but they also do not vote much for the new top labels. It seems then that the effect of the attack is to spread the votes of pixels throughout all $1,000$ classes. This hypothesis is consistent with the effect on the contribution histograms. In Figure 1 we observe that the attack has a strong effect on the variance of the contributions per layer. So for each image we get a different histogram, with strong positive and negative contributions. **The network does not behave normal with images corrupted with adversarial attacks**. The layers do not contribute in the same way and pixel votes do not show strong agreements.

B. Super–Resolution (SR)

Projective/receptive filters:

Supplementary material in section D includes a live demonstration, showing rows and columns of the linear interpreter for the upscaling methods: Bicubic, 4–layers PixelShuffle[3], and EDSR[3]. In Figure 4 we show snapshots of the demonstration. The filter matrix $F(x_0)$ for SR methods is not square and has a vertical shape. For every pixel in the input domain (small resolution) there is a column, representing the projective filter. We implement and analyze a Bicubic upscaler for two reasons: first, it helps to verify the implementation of our analysis; and second, to take it as a reference for interpretation. The demonstration let users move around an image and inspect all the filter matrix's rows and columns. It is the equivalent to materialize the matrix $F(x_0)$, except that we do not keep the matrix in memory. For these examples we precomputed all rows and columns and save them as image files. For the largest and slowest model, EDSR, we can compute more than 2 rows and column images per second on a Titan X GPU (12GB). Then, we use a modern browser that displays the diagram and loads the row/column images corresponding to the location in the image.

For SR methods we observe that **filter coefficients are sparse** since the image outside the zooming window is

[2]Attack implemented by using code from https://github.com/baidu/AdvBox.

[3]Model obtained by running a PyTorch tutorial from https://github.com/pytorch/examples/tree/master/super_resolution

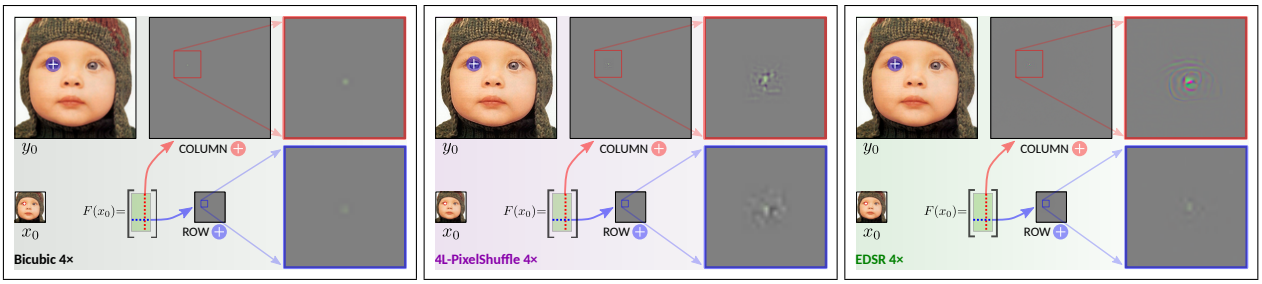

Figure 4: Screenshots of live demonstration showing the projective and receptive filters (columns and rows) for the linear interpreter of upscaler systems. A bicubic upscaler does not adapt to the image and keeps the filter coefficients unchanged. The 4–layer PixelShuffle model adapts to the image, changing on edges and textures, but does not clearly follow the geometry. The EDSR[3] model adapts to the image and reveals the geometry of high–level features (e.g. eyes, textures, nose).

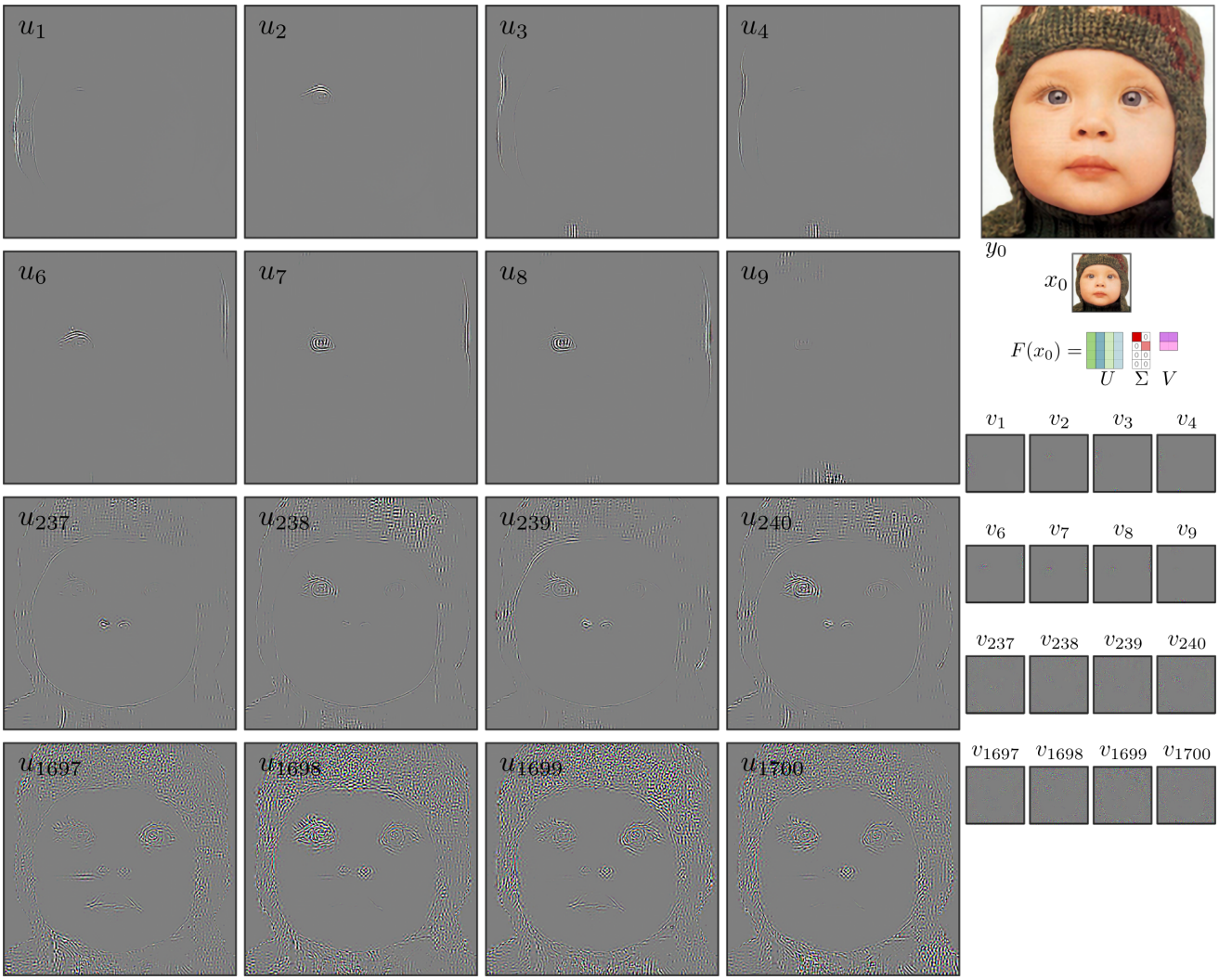

Figure 5: Results of the Singular Value decomposition of a linear interpreter applied on EDSR[3] $4\times$ super–resolution method. The basis used by EDSR is spatially localized, oriented and bandpass, comparable to wavelet basis[5], and similar to the receptive fields of simple cells in mammalian primary visual cortex. The Eigen–inputs/outputs for largest singular values capture the objects with largest receptive fields, indicating strong knowledge of the geometry of the image.

mostly full of zeros. The **coefficients are concentrated around the pixel location**, as expected, since interpolation must give preference to the current pixel location and use its neighbors to improve it.

The demonstration shows that for good models, like EDSR, a user can guess the location in the image just by looking at the projective filters (columns). In layman's terms:

> ***Inspecting SR projective filter coefficients feels like walking through the image with a flashlight.***

This observation offers a simple check to verify that the model has learned the geometry of images. On the other end, bicubic would make a user feel blind since it is completely space invariant; and the 4–layers PixelShuffle model would make users feel confused on the location because the geometry is not clearly revealed in the projective filters.

More eigen–inputs/outputs:

The images of eigen–input/outputs in the main text contain zooms that cover certain areas of the image, often full of zeros. In Figure 5 we show more eigen–input/outputs without zooming. Here, we observe that eigen–input/outputs are sparse and capture few or single features of the image (e.g. left eye, right eye, etc.) for the largest singular values. The images of eigen–input/outputs for larger singular values contain higher–frequencies and cover larger areas.

C. Image–to–Image Translation (I2I)

Projective/receptive filters:

Supplementary material in section D includes a live demonstration, showing rows and columns of the linear interpreter for CycleGAN[7] architecture and different pre–trained models[4]. In Figure 6 we show some snapshots of the live demonstration.

The filter matrix $F(x_0)$ for I2I methods is square. Here, we observe that **filters coefficients in I2I are less sparse than those in SR methods**. In areas where the content does not show strong changes we observe delta-type filter coefficients centered in the diagonal. In areas where the content is converted to a cartoon–style flat color (e.g. blue background in Photo–to–Label) the input is largely ignored or spread around a large area. In other areas the coefficients are strong around the diagonal but often include strong off–diagonal components (see Figure 6). We observe strong off–diagonal components in the columns, suggesting that the network is using both pixel values in current location, as well as values from other regions of the input image, in order to obtain the output. We also see strong off–diagonal components in the rows, suggesting that the network is using the results of the current location somewhere else in the image. The CycleGAN[7] architecture can achieve this easily by using instance–normalization layers that make use of global features (image mean and variance).

As opposed to good SR methods, when using painting styles (VanGogh and Ukiyoe) the projective filters (columns) do not follow the geometry of the image and do not easily reveal the location in the image. In the case of the Photo–to–Label model we can guess the content and location because we see windows with strong neon–style colors. **The filter coefficients in painting styles seem to focus more on textures and color**.

In the case of Photo–to–Label style, we do not observe a peak in the diagonal elements (the location of the current pixel) as seen before in SR methods and painting styles when content is preserved. Instead, we see the shape of windows turning on and off. We believe that this is caused by the nature of the problem, that is basically trying to perform segmentation. **The Photo–to–Label filter matrix works like a detection system that creates template boxes in the output**. The background blue color indicates that a segment has not been detected. The on–and–off effect suggests that a new segment has been found (e.g. an eave, a window, a door, etc). The receptive filter (rows) resembles a Gabor–like template matching filter. The projective filters (columns) show how single pixels are assigned to a whole segment in the output. For this problem, templates are simple and the network is able to create them. This is much simpler than creating templates for image classes in ImageNet where we did not observe the network following the same strategy.

Eigen–inputs/outputs:

In Figure 7 we observe how CycleGAN's eigen–decompositions show some similar patterns compared to SR models. Namely, eigen–inputs/outputs are localized for large singular values and cover larger areas for smaller singular values. Also, eigen–inputs contain high frequency stimulus that are translated into colorful textures (VanGogh and Ukiyoe styles) or template boxes (Photo–to–Label style) in their correspondent eigen–outputs.

Other patterns are clearly different. Namely, the first eigen–inputs/outputs cover larger areas than SR models, and they focus more on color, capturing some of the VanGogh style used in Figure 7. The content in eigen–outputs capture more textures, compared to SR models that focus more on curves and edges.

In Figure 8 we observe that residuals in CycleGAN models are larger than residuals observed in SR. The eigen–outputs of different styles show a clear focus of the network in generating the colors and objects of the target style. We can conclude that an SVD analysis helps to interpret a network by showing how they focus on their tasks. This is, geometric shapes for SR and texture/color styles for I2I.

[4]CycleGAN models downloaded from http://efrosgans.eecs.berkeley.edu/cyclegan/pretrained_models

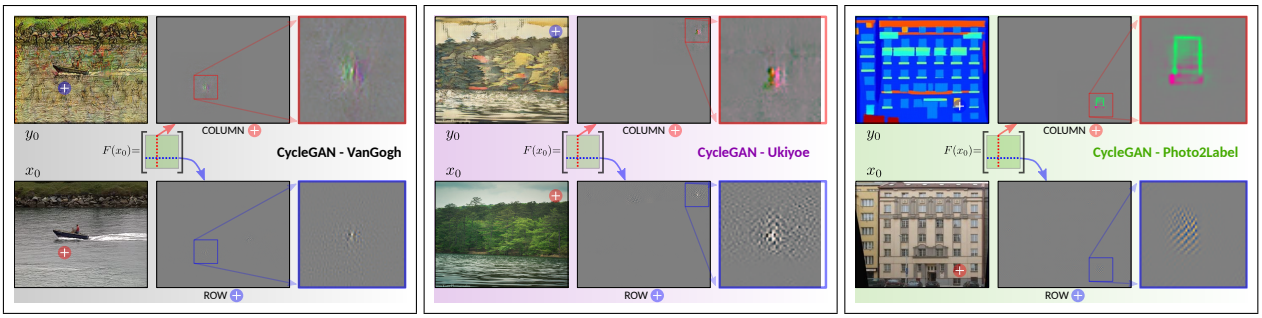

Figure 6: Screenshots of live demonstration showing columns and rows for the linear interpreter of image–to–image translation systems. The demonstration reveals strong presence of off-diagonals in the filter matrix. This means that CycleGAN chooses certain areas in a given image to copy, move and generate textures.

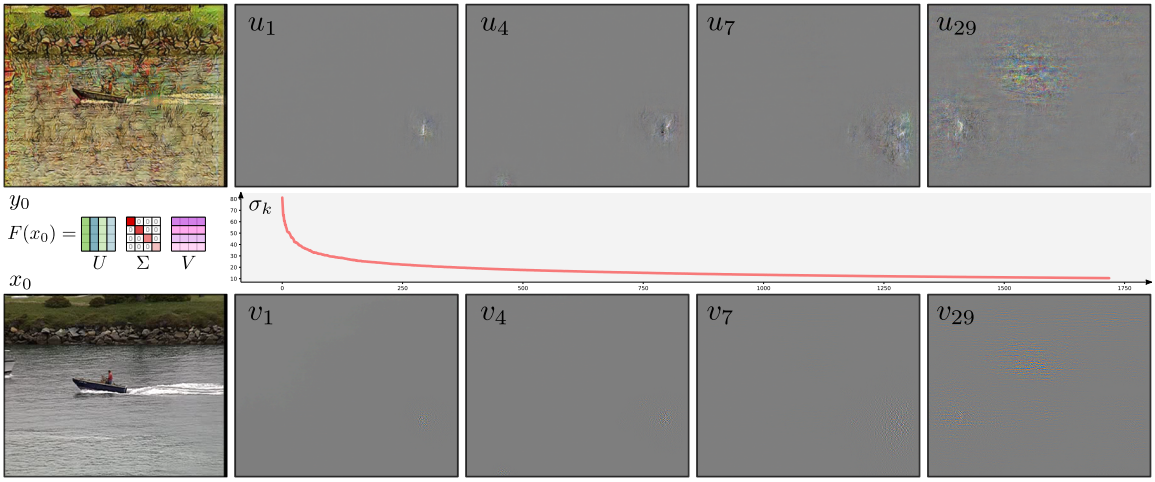

Figure 7: Results of the Singular Value decomposition of a linear interpreter applied on CycleGAN[7]–VanGogh I2I network model. Eigen–outputs for large singular values reveal the areas with largest contributions. Compared to SR eigen–decompositions, the basis is also spatially localized, oriented and bandpass, comparable to wavelet basis[5, 4]. But we observe that I2I eigen–decomposition is much less sparse than in SR, indicating a more global strategy to solve the problem.

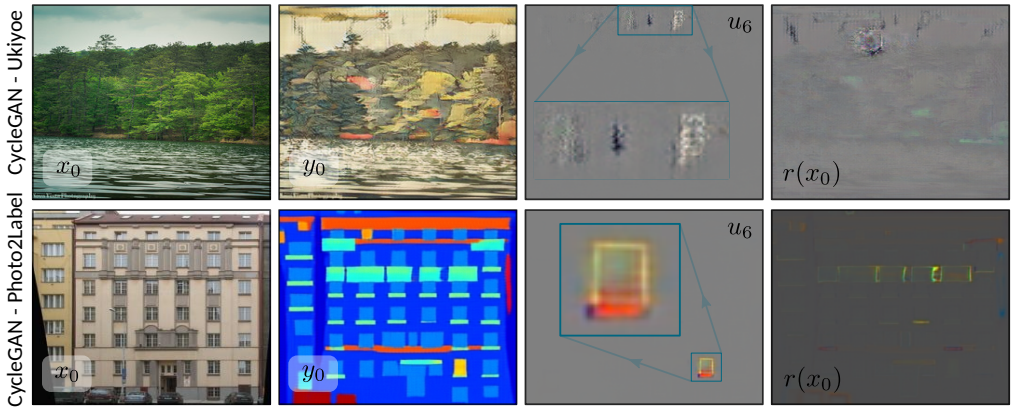

Figure 8: The SVD of I2I models shows how the network focuses on particular styles. Residuals in I2I contribute more than in SR problems. Eigen–outputs for large singular values help to identify the areas with largest contributions. In Ukiyoe style, the eigen–output u_6 shows an area originally empty in the input, where a new texture has been created. In Photo–to–Label, the eigen–output u_6 shows the creation of a template window segment.

D. Demonstrations

Video material: The following videos are included as supplementary material:

- FilterMatrix_SR.mp4
- FilterMatrix_I2I.mp4

Both videos include an English subtitle track embedded in the MP4 containers. We hope that these comments can help viewers to better understand the results of the analysis. The subtitle's font and size are controlled by the video player and can sometimes obstruct information in the video frames. Please feel free to enable/disable the subtitle track to better appreciate the results of the demonstration.

Interactive material: The interactive demonstrations can be downloaded from:

- Bicubic4x.zip (17 MB)
- 4L-PixelShuffle.zip (17 MB)
- EDSR4x.zip (953 MB)
- CycleGAN-VanGogh (7.5 GB)
- CycleGAN-Ukiyoe (4.4 GB)
- CycleGAN-Photo2Label (4.0 GB)

Please note that large file sizes (mostly I2I) are due to the fact that we recorded all rows and columns using lossless compression to avoid misinterpretations.

References

[1] Sebastian Bach, Alexander Binder, Grégoire Montavon, Frederick Klauschen, Klaus-Robert Müller, and Wojciech Samek. On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation. *PloS one*, 10(7):e0130140, 2015. 2

[2] Ian Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. In *International Conference on Learning Representations*, 2015. 3, 5

[3] Bee Lim, Sanghyun Son, Heewon Kim, Seungjun Nah, and Kyoung Mu Lee. Enhanced deep residual networks for single image super–resolution. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, July 2017. 5, 6

[4] Stéphane Mallat. *A Wavelet Tour of Signal Processing*. Academic Press, 1998. 8

[5] Bruno A Olshausen and David J Field. Emergence of simple–cell receptive field properties by learning a sparse code for natural images. *Nature*, 381(6583):607–609, 1996. 6, 8

[6] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015. 2

[7] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A Efros. Unpaired image–to–image translation using cycle-consistent adversarial networks. *arXiv preprint arXiv:1703.10593*, 2017. 7, 8