A Tour of Convolutional Networks Guided by Linear Interpreters

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

Abstract

Convolutional networks are large linear systems divided into layers and connected by non–linear units. These units are the "articulations" that allow the network to adapt to the input. To understand how a network manages to solve a problem we must look at the articulated decisions in entirety. If we could capture the actions of non–linear units for a particular input, we would be able to replay the whole system back and forth as if it was always linear. It would also reveal the actions of non–linearities because the resulting linear system, a ***Linear Interpreter****, depends on the input image. We introduce a hooking layer, called a* ***LinearScope****, which allows us to run the network and the linear interpreter in parallel. Its implementation is simple, flexible and efficient. From here we can make many curious inquiries: how do these linear systems look like? When the rows and columns of the transformation matrix are images, how do they look like? What type of basis do these linear transformations rely on? The answers depend on the problems presented, through which we take a tour to some popular architectures used for classification, super–resolution (SR) and image–to–image translation (I2I). For classification we observe that popular networks use a pixel–wise vote per class strategy and heavily rely on bias parameters. For SR and I2I we find that CNNs use wavelet–type basis similar to the human visual system. For I2I we reveal copy–move and template–creation strategies to generate outputs.*

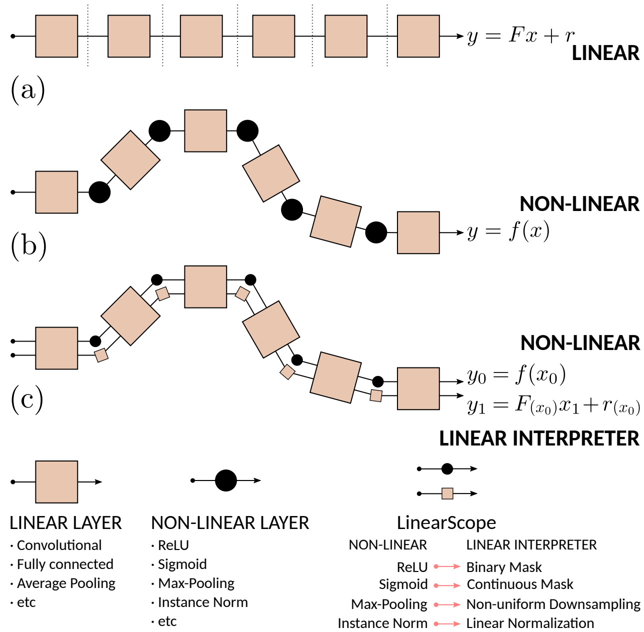

Figure 1: (a) Attaching linear layers of a network gives a linear system. (b) Non–linear units work as "articulations" that make the network adaptive to the input. (c) We can run the network in two batches, and use a LinearScope in each non–linear unit to run the network on the first batch, and a linear interpretation of the non–linear action in the second batch. The output in the first batch is unaffected by LinearScopes. The second batch gives a linear interpreter of the whole network that depends non–linearly on the first batch and linearly on the second batch.

1. Introduction

In this paper we are going to explore the interpretability of convolutional networks by using linear systems. The main task is to improve our understanding of how deep neural networks solve problems. This has become more intriguing because of the predominance of deep–learning systems in machine learning benchmarks, which has motivated extensive research in this area. The question is how to *interpret* the models, and the interpretation can reflect many different ideas[23]. But before we get into the meaning of interpretability, let us first remember that the design of neural networks was simple from its very beginning: linear systems and (non–linear) *activations*[36]. Here, *activations* are biologically inspired to refer to inhibition of features (the output of the linear system), and the usual circuitry analogy is a switch. The problem arise when we combine many of these simple units, run many features in parallel, and subsequently repeating the same process. More precisely, it is not clear how the partial results lead us to the final decision.

Linear systems are generally considered *interpretable* given a long history of research[43]. With a linear system we know what to expect and where to look at to find answers. Here, we are interested in some of their most important properties. We will write an affine transformation $y : \mathbb{R}^n \to \mathbb{R}^N$ as

$$y(x) = Fx + r \quad (1)$$

where $r \in \mathbb{R}^N$ is a **residual** that lives in the same space as the output, and it is thus visible and interpretable as a fixed shift. The next useful information comes directly from the **rows** and **columns** of the matrix F. A row shows us the input pixels that are used to get an output pixel. We call these the *receptive filter* coefficients as their extension in space show the *receptive field* of the model. On the other hand, a column shows us the output pixels affected by an input pixel. We call them the *projective filter* coefficients and their extension in space the *projective field* of the model. Other important information comes from the **transposed system**, represented by F^T, which interchanges the meaning of rows and columns and *back*–projects vectors from the output domain back to the input domain.

To interpret a linear transformation Fx as a whole, which is to get a feeling of what parts of an input signal passes and how much it passes, we need its **singular value decomposition** (SVD), $F = U\Sigma V$. This gives us a full description of the vector spaces connecting input and output domains. The set of left (U) and right (V) eigenvectors basically shows us what the outputs and inputs are made of according to the transformation. For linear space invariant systems (LSI)[33], these are harmonic functions like complex exponentials $U_{jk} = e^{-i\Omega jk}$ or some type of DCT[44]. These systems play a fundamental role in signal processing[33, 24]. In simple terms, in LSI systems waves move in and out without changing their shapes, and can be interpreted as the natural choice of the system to decompose inputs and outputs. When a matrix is not symmetric or square, left and right eigenvectors are different. For the sake of simplicity, we will call them **eigen–inputs** and **eigen–outputs**, so that they remind us of the space where they live. What matters here is a pair of eigen–input, $v \in \mathbb{R}^n$, and eigen–output, $u \in \mathbb{R}^N$. An eigen–input transformed with F gives us an eigen–output rescaled by its singular value σ, $u = \sigma F v$, and the eigen–output back–projected with F^T returns the rescaled eigen–input, $v = \sigma F^T u$. So in general terms, a pair of eigen–input/output moves in and out, projected and back–projected, without changing their shapes, just rescaled by their singular values. The singular value shows the filtering effect, which represents what passes and how much passes. A small singular value indicates a pair of eigen–input/output that vanishes quickly after a transformation and back–projection.

Now, why should we use linear systems to interpret convolutional networks? We cannot study a structure made of material A by using our knowledge on material B, just because we know B better. Linearizations of convolutional networks can indeed be very useful, and have been studied in [25] to obtain heatmappings that show the relevance of inputs in the outputs. Its connections with our results will be discussed later. Here, we want to emphasize two simple arguments as to why should we use linear systems:

1. **Convolutional networks are largely made of linear systems**. In fact, all the parameters of a network are contained in linear modules (e.g. convolutional layers) with few exceptions (e.g. Parametric ReLU);

2. **The design of non–linear units have an initial linear motivation**, and the non–linearity is added in order to select their linear parameters adaptive to the input. Activations like ReLU or Sigmoid are switches that can be represented by pixel–wise masks multiplying inputs. If we fix the mask, it becomes linear. A max–pooling layer selects one among a group of pixels and allows a similar interpretation by using selection masks. An instance–normalization layer subtracts a mean and divides by a standard deviation. If we fix the mean and standard deviation, it becomes linear. Now, we do have simple linear interpretations of non–linear units.

So, if we use the linear interpretation of non–linear layers (meaning to freeze the decisions of non–linear units), the whole system becomes linear. This procedure has been used in [28] to visualize how CNNs upscale small images. The authors proposed to replace activation units by masks and thus obtained linear systems of the form $y = Fx + r$. By inspecting the columns of F, they observed upscaling coefficients highly–adaptive to the input.

This work focuses on experimental explorations. Similar to a laboratory that needs a microscope to study microorganisms, we need an instrument to perform studies with linear interpreters. Thus, a key contribution is the design of a hooking layer (LinearScope), that can be inserted in CNNs to extract information. With this tool in hand we are able to extend an existing approach of interpretability[28] to significantly broader applications, through which we have made the following important discoveries:

- We report a **"pixel–wise vote" interpretation of image classifiers** in which each pixel votes independently for an image label and the most voted label gives the output. Other works have found that classification CNNs are biased towards textures[17], or that they still perform well after shuffling patches[20], while our results point to the concrete strategy of the network (pixel votes).

- We report a **critical role of the bias parameters in CNNs for image classification**, as opposed to other applications (e.g. SR and I2I). Moreover, they become more relevant in architectures with better benchmarks and, in the case of sequential networks we find the contributions to concentrate on specific layers that move deeper when trained with batch normalization.

- We explain the **strategies of CycleGAN to solve I2I**. We uncover a copy–move strategy for photo–to–painting task (moving textures from different places in an image) and a template–creation strategy for the facades–to–label task. It should be noted that prior to this paper, it was largely unknown how to identify the source of newly generated objects and textures.
- We derive an algorithm using LinearScopes to **obtain the SVD of a linear interpreter**. This shows us the basis behind a CNN. Here, we found strong connections to the Human Visual System (HSV). It is known that the receptive fields of simple cells in mammalian primary visual cortex can be characterized as being spatially localized, oriented and bandpass, comparable to wavelet basis. In [31] it is shown that a coding strategy that maximizes sparseness is sufficient to account for these properties, and have been of great impact in the field of sparse coding. Our SVD results reveal that the basis used by SR and I2I networks also contain all three properties above. In terms of output knowledge, it gives us an overview of the strategy to map input to output pixels.

These results may bring about the following **future impact**: 1) the explicit demonstration that CNNs use wavelet–type basis similar to the human visual system, 2) the creation of tools to visualize and fix problems in CNN architectures, and 3) the possibility to use the filter/residual in a loss function and design CNNs with an interpretable target.

2. Related Work

The interpretability of convolutional networks is closely related to visualization techniques. Visualization is more generally concerned on visual evidence of information learned by a network[29]. Interpretability tries to explain the inner processing of a network, and each interpretation comes with a visualization technique that we can use to interpret the learning process. Reviews of the extensive literature in visualization can be found in [49, 34, 30, 29].

The meaning, or many meanings, of interpretability is a subject of study. In [23], for example, authors identify a discordant meaning of interpretability in existing research and discuss the feasibility and desirability of different notions. They also emphasize an important misconception, that linear models are not strictly more interpretable than deep neural networks. In [13], authors define interpretability relative to a target model and not as an absolute concept. In [1], authors show how assessments relying only on the visual appealing of saliency methods can be misleading and they propose a methodology to evaluate the explanations that a given method can provide. Finally, in [18] authors show how the interpretation of neural networks is a fragile process, showing how they can introduce small perturbations in images leading to very different interpretations.

Extensive work has been done to explain the decisions of image classifiers and segmentation[12, 35, 40, 27, 3, 11, 15, 12, 35, 40, 27, 3, 11, 15, 37]. Other research directions on image classification try to find answers inside a network architecture. In [10], for example, authors study invariances in the responses of hidden units and find that these are major computational component learned by networks. In [16], authors study the collaboration of filters to solve a problem and find that multiple filters are often required to code a concept, and single filters are not concept specific. In [21], authors show that the last layer of a network works as a linear classifier, similar to the motivation of the perceptron[36].

An important research direction is to study the role of semantics. The Network–Dissection framework has been proposed in [4] to quantify the interpretability of latent representations by evaluating the alignment between individual hidden units and a set of semantic concepts. In [50], a new framework is proposed to decompose activations of the input image into semantically interpretable components. And the GAN–Dissection framework has been proposed for visualizing the structure learned by generative networks[5].

Our interpretation of CNN–classifiers are more closely related to: **Layer–wise Relevance Propagation (LRP)**[2, 6] and **Deep Taylor Decomposition** (DTD)[25]. LRP is the first framework to introduce a general solution to the problem of understanding classification decisions by pixel–wise decomposition of network classifiers, and DTD is the first study to consider Taylor decompositions in a network. The relation to our results will be discussed in Section 5.

Finally, our analysis is an extension of **Deep Filter Visualization (DFV)**, introduced in [28] to visualize how convolutional networks upscale low–resolution images. DFV proposes to replace activation units by masks and thus obtains a linear system of the form $y = Fx + r$. DFV has been used to inspect the columns of F and observe upscaling coefficients highly–adaptive to the input. In DFV one needs to record the activations for every non–linear unit in order to run the linear interpreter. This comes with a high storage cost for common architectures as shown in Table 1. If we do not have enough memory in a device (e.g. GPU), we need to switch to slower storage such as CPU DRAM, SSD or HDD with an overwhelming cost in speed, as shown in Table 2. We propose a solution to this problem that does not require to store activations, and instead requires an additional batch in the input. This novel approach gives us a much simpler and efficient implementation of the linear interpreter. We are not only able to run faster and use larger images, but we can also perform more complex analysis on the linear inter-

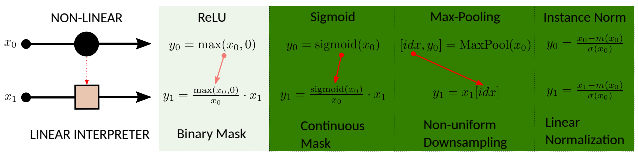

Figure 2: A LinearScope keeps a non–linear unit unchanged on batch x_0 and adds a second batch x_1 to run a linear interpreter. Red lines show how the interpreter looks at the first batch to decide: what mask to use (ReLU and Sigmoid), what inputs to select (MaxPooling), or what normalization mean and variance to use (Instance Normalization).

Network	VGG–19[41]	CycleGAN[51]	EDSR[22]
Space	58 GB	90 GB	4,147 GB

Table 1: Storage space needed to store all ReLU activations.

Storage	GPU	CPU	SSD	HDD
Speed	100%	50%	0.5%	0.005%

Table 2: Relative speed of typical storage media, taking as reference GPU (DDR5 or HBM2).

preter, including: transposed linear interpreters and singular value decompositions. State–of–the–arts CNNs are often pushed to the limit of current technologies which makes our solution critical for experimental explorations with a $2\times$ to $10^4\times$ speedup over DFV[28] according to Tables 1 and 2.

3. The Linear Interpreter

LinearScopes: We define a LinearScope as a hooking layer that modifies a non–linear unit by adding an additional batch. If a non–linear unit calculates $y_0 = h(x_0)$ on a batch x_0, then we change it to calculate:

$$[y_0, y_1] = [h(x_0), A(x_0)\ x_1 + c(x_0)]\ . \tag{2}$$

Here, $[\cdot,\cdot]$ denotes concatenation in the batch dimension, and $A(x_0), c(x_0)$ are chosen depending on our interpretation of $h(x_0)$. A hard requirement is

$$x_0 = x_1 \quad \Rightarrow \quad y_1 = y_0\ . \tag{3}$$

One choice of linear interpreter is the best linear approximation of h given by the Taylor expansion around the input:

$$\begin{aligned} h(x_1) &= h(x_0) + (Dh)(x_0)\cdot(x_1 - x_0) + \cdots \\ &= \underbrace{(Dh)(x_0)}_{A(x_0)} \cdot x_1 + \underbrace{h(x_0) - (Dh)(x_0)\cdot x_0}_{c(x_0)} + \cdots \end{aligned} \tag{4}$$

so that $y_1 = A(x_0)\ x_1 + c(x_0)$ is the Taylor interpreter.

Here, we follow and extend the approach of DFV[28], which is not to seek an approximation. We prefer to use the word *freezing* instead of *linearization*. We think of the DFV approach as follows: the network has taken some decisions throughout its layers for an input image (See Figure 1). Figure 2 shows the unique choices to fix these decisions. The overall *frozen* system happens to be linear because of the particular structure of CNNs, as opposed to a Taylor expansion that forces linearity in the interpreter.

Linear Interpreter: Figure 1 explains our general idea. We want to use the LinearScope hooking layers inside a model to replace all its non–linear units. If a network outputs $y_0 = f(x_0)$, with $x_0 \in \mathbb{R}^n$ and $y_0 \in \mathbb{R}^N$, then a model with LinearScopes outputs:

$$[y_0, y_1] = [f(x_0), F(x_0)\ x_1 + r(x_0)]\ , \tag{5}$$

where $F(x_0) \in \mathbb{R}^{N\times n}$ is the *filter* matrix and $r(x_0) \in \mathbb{R}^N$ is the *residual*. A key idea proposed in DFV[28] is that we do not need to materialize the matrix $F(x_0) \in \mathbb{R}^{N\times n}$ to run the linear interpreter. The model with LinearScopes also avoids storage of activations in non–linear units because this information is used on–the–fly within LinearScopes (red lines in Figure 2) and it is released afterwards.

Finally, our purpose will be to fix an input image x_0 and run tests with different probe inputs x_1 to get information from the linear interpreter.

Residual and Columns: The procedure to calculate the residual $r(x_0)$ and columns of $F(x_0)$ from the linear interpreter follows the solution from DFV[28]. The residual is given by $y_1 = r(x_0)$ when we use a probe batch $x_1 = 0$. Next, we can obtain a column k from the filter matrix $F(x_0)$ as $y_1 - r(x_0)$ when we use a probe batch $x_1 = \delta_k$, where $\delta_k[k] = 1$ and $\delta_k[i \neq k] = 0$. This is an impulse response function according to signal processing theory[33, 24].

Transposed System and Rows: To calculate $F^T(x_0)\cdot y_2$ for a given image in the output domain, $y_2 \in \mathbb{R}^N$, we can use the vector calculus property for gradients of linear transformations: $\nabla_x(Ax+b)y = A^T y$. The same approach is used to implement (strided) transposed convolutions in deep learning frameworks[32], except that here our system is much bigger (possibly including transpose convolutions). Since deep learning frameworks provide automatic differentiation packages, it is simple and convenient to calculate:

$$F^T(x_0)\cdot y_2 = \nabla_{x_1} y_1(x_1)\cdot y_2\ . \tag{6}$$

Finally, we can use the impulse response approach to obtain the rows of $F(x_0)$. This is, a row k from the filter matrix $F(x_0)$ is given by $F^T(x_0)\cdot\delta_k$ when we use a probe image $y_2 = \delta_k$, where $\delta_k[k] = 1$ and $\delta_k[i \neq k] = 0$.

Before moving forward, we emphasize that the transposed linear interpreter is different than the popular deconvolution method by Zeiler et.al[48] because the deconvolution uses a non–linear output. More precisely, the procedure in [48] describes how each layer must be transposed.

The linear interpreter follows the same procedure for convolutional layers (linear) and max–pooling (our linear interpreter is equivalent to their approach), but for ReLU the approach in [48] is to use an identical ReLU unit (non–linear). Instead, the linear interpreter will remember the activation of the unit in the forward pass (through gradients) and use the masking interpretation (linear).

Singular Value Decomposition (SVD): The dimension of inputs $x \in \mathbb{R}^n$ and outputs $y \in \mathbb{R}^N$ of a network can be different. Then the eigendecomposition of the filter matrix is given by its singular value decomposition (SVD). We propose Algorithm 1 to calculate the eigen–input/output for the largest singular value of $F(x_0)$, without materializing the matrix. We use an accelerated power method with momentum[47] adapted for SVD[7]. Further eigen–inputs/outputs can be calculated in decreasing order of singular values by using a deflation method[9, 39]. For example, the second eigen–inputs/outputs and singular value is calculated by using Algorithm 1 on the *deflated system* $F(x_0) + r(x_0) - \sigma_1 u_1 v_1^T$, and so forth.

Algorithm 1 SVD power method for a Linear Interpreter

Input: Test image x_0.
Input: Linear interpreter $y_1(x_1|x_0)$.
Input: Residual $r(x_0)$.
Input: Momentum m, number of steps S.
Outputs: σ_{curr}, v_{curr}, u.

$m \leftarrow 0$, $\sigma^2_{prev} \leftarrow 0$, $v_{prev} \leftarrow 0$, $v_{curr} \leftarrow \mathcal{N}(0,1)$
for $it = 1, \ldots, S$ **do**
  $u \leftarrow y_1(v_{curr}|x_0) - r(x_0)$
  $v_{next} \leftarrow F^T(x_0) \cdot u - m * v_{prev}$ use equation (6)
  $\sigma^2_{curr} \leftarrow v^T_{curr} \cdot v_{next}$
  $v_{prev} \leftarrow v_{curr}/||v_{next}||$
  $v_{curr} \leftarrow v_{next}/||v_{next}||$
  $\sigma^2_{prev} \leftarrow \sigma^2_{curr}$
end for
$u \leftarrow u/||u||$

4. Experiments

Case 1 – Classification: In this case a network takes images into scores (we do not include a softmax layer). If we look at a single score for a test image x_0 then $F(x_0) \in \mathbb{R}^{1 \times n}$ is a single row image. Here, we are tempted to make a guess. We have seen evidence in DFV[28] that residuals are small. Then, if we want to maximize $F(x_0)x_0$ an ideal choice would be template–matching[8, 45]. This is, the network could try to construct a template image $F(x_0)$ that looks similar to x_0 for the correct label. In our experiments with various architectures we find that this is not the case. The image $F(x_0)$ does not look like a template and, most importantly, the residual $r(x_0)$ has the largest contribution

AlexNet	VGG–19	ResNet–152
78.5%	85.5%	81.1%
SqueezeNet 1.1	DenseNet–161	Inception v3
84.3%	95.0%	91.6%

Table 3: Average contributions of residuals for 100 validation images from ImageNet–$1k$[38]. The percentage increases for architectures with better benchmarks.

to the scores, typically adding more than 80% of the contribution as shown in Table 3. This is a discouraging fact to conduct analysis since the residual of a score is a scalar that does not give more information than the score itself.

But additional information can be obtained by using a theorem for sequential networks. For the sequential model:

$$y_n = W_n x_{n-1} + b_n \qquad \text{and} \qquad x_n = h\left(y_n\right) \ , \tag{7}$$

with parameters b_n (biases) and sparse matrices W_n (convolutions), we can get explicit formulas for the filter matrix and residual. This is:

Theorem 1 (from [28]) *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 of $h(y_n)$. Let $Q_n = I$ and $Q_i = \prod_{k=i+1}^{n} \hat{W}_k$ for $i = 1, \ldots, n$. The filter matrix and residual are:*

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

Let us grasp the meaning of this result. We will focus on networks with ReLU units so that $c_n = 0$. First, the parameters with hat, $\hat{W}_n$ and $\hat{b}_n$, are the weights and biases of the network multiplied by masks. This already depends on the test image x_0. So, the formula for F in (8) basically represents the accumulated convolutions, masked by activations.

Next, matrices Q_i represent the accumulated effect of convolutions, masked, from layer $i + 1$ to n (a forward projection). So finally, the formula for r in (8) gives us a **decomposition of the residual as layer–wise contributions of biases**, masked and forward projected into the scores.

In Figure 4 we show a histogram of the contributions for top–1 scores in a pre–trained VGG–19 network[41], averaged over 100 validation images from ImageNet–$1k$[38]. This includes a contribution of the input $F(x_0)x_0$ and the layer–wise contributions of the masked biases. We observe that most contributions come from the first two layers (with high variance) and the three layers before the fully–connected layers. For other variants of VGG we consistently observe two main contributions: one peak in early layers, and a second peak right before fully connected layers. But when the network is trained with batch normalization, the contributions move deeper in the network with one major contribution right before fully connected layers

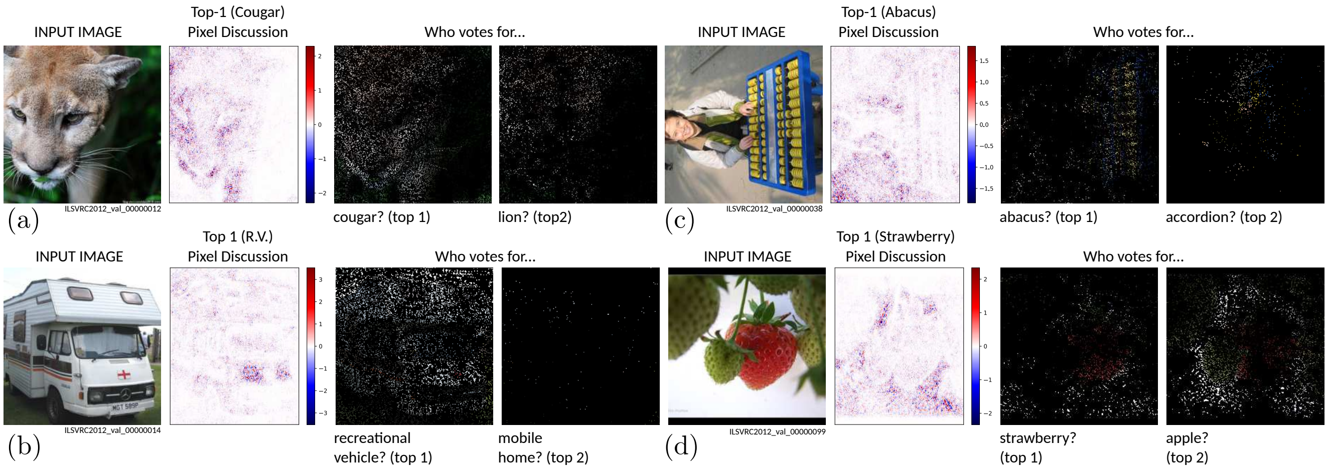

Figure 3: We back–project all the score contributions to input domain to show pixel–wise contributions, called *pixel discussions* because pixels do not seem to agree on the scores. By comparing contributions among all scores, we make pixels vote independently and find that they finally focus on objects, with top–2 scores that show reasonable arguments for their votes.

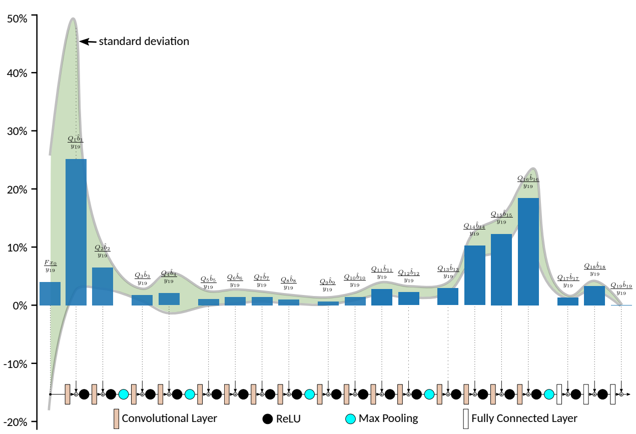

Figure 4: Layer–wise contributions to Top–1 scores for VGG–19 classifier[41], averaged over 100 images from ImageNet–$1k$[38] and normalized by the output score.

(see appendix). Early contributions are based on local information as opposed to late contributions that use global information. This is reminiscent of results in [26] (section G) that use a similar linear mapping interpretation, discovering that hidden units learn to be invariant to more abstract translations at higher layers. In the appendix we also show how the contributions inside a network become random for images corrupted with adversarial noise using FGSM[19], and final scores are exclusively due to the first few layers.

We can also perform a backward analysis by taking all the masked biases and back–project them from each layer to the input domain, adding them to $F(x_0)x_0$. We can perform this computation by considering subsystems from the input to an intermediate layer k and use F_k^T (using equation (6)) on the masked biases $\hat{b}_k$. By summing all the back–projected contributions, we can see the pixel–wise contributions for each score. Examples are shown in Figure 3 for top–1 scores (more details in the appendix). We call these images **pixel discussions** because of the random behavior of pixels. They do not represent heatmaps because: first, highest values do not always focus on the objects; and second, positive values are followed by negative values in almost every pixel, as if pixels always digress with their neighbors on the contributions to the score. It should be noted that similar images are observed in LRP studies[2, 6].

Finally, we uncover clear information after we take each pixel contribution and compare it to the same pixel contributions for all other labels. In this way, **we make each pixel vote for a label**. In Figure 3 we mask the test image using the votes per pixel to observe what areas are more popular among pixels for a given label. The top–1 scores normally show the largest popularity and, most importantly, pixels clearly focus on objects. In Figure 3 (a) and (b), for example, pixels seem to discuss randomly on the face of a cougar and the lights of a vehicle, but when it comes to votes then distinctive features of the cougar appear as well as the whole vehicle. The votes for lion on 3 (a) show areas that could actually look more like a lion, so these pixels seem to have an argument. In Figure 3 (c) and (d), pixels discuss randomly in areas that do not contain the main object, but after voting they do focus on the objects. Figure 3 (d) is interesting because the votes for strawberry show the red shape of a strawberry, and the votes for apple do show green and red shapes that resemble a couple of apples.

Case 2 – Super–Resolution (SR): In this case a network takes small images into large images. The filter matrix $F(x_0) \in \mathbb{R}^{N \times n}$, with $N > n$, has a tall rectangular shape. The linear interpreter analysis was originally used

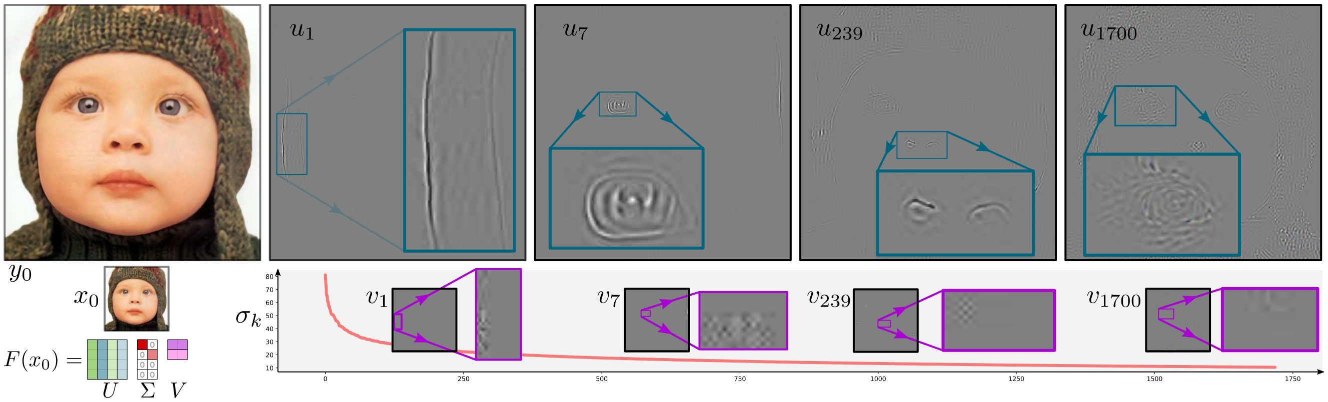

Figure 5: Results of the SVD of a linear interpreter applied on EDSR[22] $4\times$ super–resolution method.

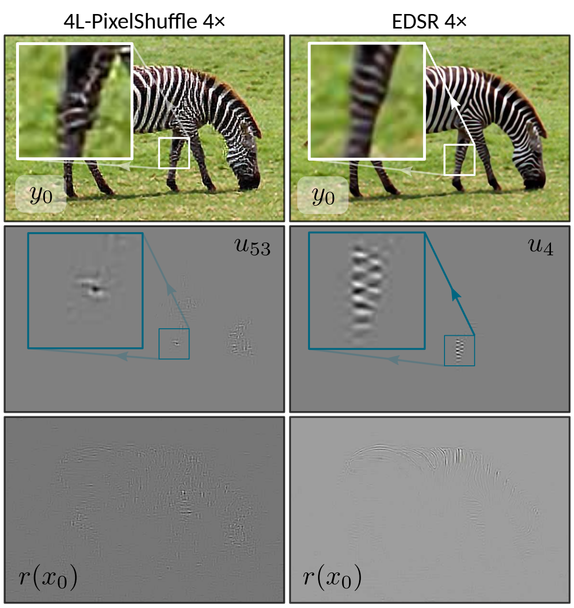

Figure 6: The SVD of SR models show how better models (EDSR) capture higher–level features from images.

in DFV[28] to study this problem. In [28] only projective filter coefficients were obtained (columns of the filter matrix). We show results with receptive filter coefficients in the appendix, which are more closely related to the traditional concept of convolutional filters. In addition, we can now efficiently calculate all the rows and columns for a given image, using very big models such as EDSR[22] (see demonstrations in the appendix).

Figure 5 shows examples of the eigen–inputs/outputs and singular values of EDSR[22] $4\times$ upscaler. Before we interpret these results it is convenient to remember a simple reference. A classic upscaler uses linear–space–invariant (LSI) filters[33, 24] whose eigen–inputs/outputs are harmonic functions (e.g. some type of DCT). So, our reference from classic upscaling are basis that cover all the image using different frequencies. The information in Figure 5 reveals a very different approach followed by convolutional networks. First, we observe oscillations of high frequencies in the eigen–inputs. These are similar to high frequency stimulus used in psychovisual experiments of contrast sensitivity function, where subjects are required to view sequential simple stimuli, like sine–wave gratings or Gabor patches[46]. The response of the network to these stimuli are clear pieces of images (e.g. an eye, a corner, a nose, etc.), smooth and localized in space for high singular values, and extending in space with higher frequency components for lower singular values. So the network reacts to stimulus similar to Gabor wavelets by triggering image objects. The response is similar to the receptive fields of simple cells in mammalian primary visual cortex that can be characterized as being spatially localized, oriented and bandpass, comparable to wavelet basis[31, 24]. Compared to EigenFaces obtained by PCA decompositions[42], we observe a similar pattern of low to high frequency oscillations as the eigen/singular–values reduce. But EigenFaces are not localized like the CNN eigen–decomposition in Figure 5.

Finally, in Figure 6 we show how an SVD analysis helps to evaluate models. A 4–layer PixelShuffle model commonly used in deep–learning tutorials is compared to EDSR model. The image quality of EDSR is clearly better. We observe that residuals are small for SR models. For EDSR the residual is more focused on the back and neck of the zebra, whereas the residual in PixelShuffle is spread all over the image. In the eigen–outputs we see that EDSR focuses in features that are visible parts of the zebra. The eigen–output where the PixelShuffle model focuses on the same area (back leg), does not show clear local features of the zebra. We can conclude that better models are able to capture and focus on high–level objects in an image.

Case 3 – Image–to–Image Translation (I2I): We end our tour with a network that does not change the size of images. The filter matrix $F(x_0) \in \mathbb{R}^{N\times n}$, with $N = n$, is

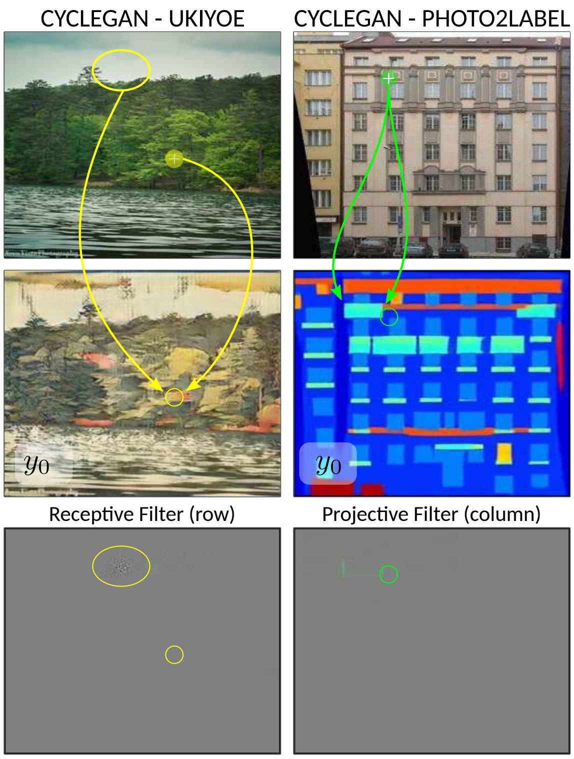

Figure 7: Receptive and Projective filters of the linear interpreter for CycleGAN[51] Ukiyoe and Facades. An off–diagonals (yellow ellipsis) is used in Ukiyoe to help generating textures. A single pixel helps to create a template window box in Facades.

square. Here, we choose to test different pre–trained models of the popular CycleGAN architecture[51]. This architecture uses instance–normalization layers that are known to improve visual effects by using global information (means and variances of network features). For this, we use the linear interpreter shown in Figure 2.

In Figure 7 we show projective and receptive filter coefficients for two I2I tasks: image–to–painting (similar to style transfer) and photo–to–facade (similar to segmentation). On one hand, compared to SR, the I2I tasks show some similarities. In most areas of an image we observe localized filter coefficients (see demonstrations in the appendix) which means that the filter matrix is sparse and concentrated around the diagonal, similar to SR. But on the other hand, the receptive/projective fields are larger in CycleGAN and the most distinctive feature is the appearance of strong off–diagonals. Figure 7 shows how in photo–to–painting the receptive filter uses information localized to a particular output location (small circle) and adds significant information from an area in the upper part of the image (the ellipsis). We observe that for a single image, CycleGAN consistently uses the same area (e.g. the ellipsis in Figure 7) to pass information to all other pixels in the image. This **copy–move strategy** seems to give the ability to create a consistent texture all over the image, taken from a fixed place and combined with the local pixels.

In the photo–to–facade task, besides the appearance of strong off-diagonals, we observe how single pixels are directed to specific segments of the output. By this means, CycleGAN **creates templates** (e.g. window boxes) that are usually triggered by pixels in corner or edges as shown in Figure 7. Also, for this case, the receptive filter coefficients can sometimes extend to the whole image (see demonstration in the appendix). This behavior is only possible due to instance normalization layers carrying global information of the image. In SR tasks, usually trained over relatively small patches (e.g. 48×48 in small resolution) a network cannot learn such strategies. Pretrained models of CycleGAN used whole images (256×256) for training.

Results of SVD decomposition for CycleGAN are included in the appendix. Here, the eigen–inputs/outputs show similar patters to SR but the stimuli and responses in the output cover much larger areas and show several objects in the eigen–outputs as opposed to single objects observed in SR. This is likely caused by off–diagonal patterns.

5. Discussion

LRP[2] introduces the concept of relevance of each pixel in the classification scores. If we use our layer–wise contributions to redefine LRP relevances we could force our analysis to fit into the LRP framework. Our contributions are significant because of the novel interpretation, revealing an explicit contribution of biases to the final scores that was previously unknown. At pixel level, LRP has been used to study the influence of input pixels to the final scores in order of pixel–wise relevances[6]. On the other hand, pixel–discussions can be used independent of the scores to obtain the vote of each pixel. Besides this difference, further investigation is necessary to better understand the relationship between pixel–discussions and other heatmap visualizations.

DTD[25] uses layer–wise Taylor expansions and modifies the root points to obtain heatmaps that are consistent (conservative and positive). In our analysis we do not control the backprojections leading to pixel–discussions and as a result we find that they do not work as heatmaps but as independent votes. The targets and results of interpretability compared to DTD are therefore different, but further investigation is necessary to better understand this relationship.

Finally, our approach in this paper relies on the human understanding of linear systems. Therefore, the effect of visualization results on human understanding is not direct. Future research is necessary to understand whether humans can predict model failures better, as proposed in [14], with or without access to LinearScope visualizations.

References

[1] Julius Adebayo, Justin Gilmer, Michael Muelly, Ian Goodfellow, Moritz Hardt, and Been Kim. Sanity checks for saliency maps. In *Advances in Neural Information Processing Systems*, pages 9505–9515. 2018. 3

[2] 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. 3, 6, 8

[3] Aditya Balu, Thanh V Nguyen, Apurva Kokate, Chinmay Hegde, and Soumik Sarkar. A forward-backward approach for visualizing information flow in deep networks. *arXiv preprint arXiv:1711.06221*, 2017. 3

[4] David Bau, Bolei Zhou, Aditya Khosla, Aude Oliva, and Antonio Torralba. Network dissection: Quantifying interpretability of deep visual representations. In *Computer Vision and Pattern Recognition*, 2017. 3

[5] David Bau, Jun-Yan Zhu, Hendrik Strobelt, Zhou Bolei, Joshua B. Tenenbaum, William T. Freeman, and Antonio Torralba. GAN dissection: Visualizing and understanding generative adversarial networks. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2019. 3

[6] Alexander Binder, Grégoire Montavon, Sebastian Lapuschkin, Klaus-Robert Müller, and Wojciech Samek. Layer-wise relevance propagation for neural networks with local renormalization layers. In *International Conference on Artificial Neural Networks*, pages 63–71. Springer, 2016. 3, 6, 8

[7] Avrim Blum, John Hopcroft, and Ravindran Kannan. Foundations of data science. *Vorabversion eines Lehrbuchs*, 2016. 5

[8] Roberto Brunelli. *Template matching techniques in computer vision: theory and practice*. John Wiley & Sons, 2009. 5

[9] Richard L Burden and J Douglas Faires. Numerical analysis. *Cengage Learning*, 9, 2010. 5

[10] Santiago A Cadena, Marissa A Weis, Leon A Gatys, Matthias Bethge, and Alexander S Ecker. Diverse feature visualizations reveal invariances in early layers of deep neural networks. *arXiv preprint arXiv:1807.10589*, 2018. 3

[11] Marco Carletti, Marco Godi, Maedeh Aghaei, and Marco Cristani. Understanding deep architectures by interpretable visual summaries. *arXiv preprint arXiv:1801.09103*, 2018. 3

[12] Amit Dhurandhar, Pin-Yu Chen, Ronny Luss, Chun-Chen Tu, Paishun Ting, Karthikeyan Shanmugam, and Payel Das. Explanations based on the missing: Towards contrastive explanations with pertinent negatives. *arXiv preprint arXiv:1802.07623*, 2018. 3

[13] Amit Dhurandhar, Vijay Iyengar, Ronny Luss, and Karthikeyan Shanmugam. Tip: Typifying the interpretability of procedures. *arXiv preprint arXiv:1706.02952*, 2017. 3

[14] Finale Doshi-Velez and Been Kim. A roadmap for a rigorous science of interpretability. *arXiv preprint arXiv:1702.08608*, 150, 2017. 8

[15] Mengnan Du, Ninghao Liu, Qingquan Song, and Xia Hu. Towards explanation of DNN–based prediction with guided feature inversion. *arXiv preprint arXiv:1804.00506*, 2018. 3

[16] Ruth Fong and Andrea Vedaldi. Net2vec: Quantifying and explaining how concepts are encoded by filters in deep neural networks. *arXiv preprint arXiv:1801.03454*, 2018. 3

[17] Robert Geirhos, Patricia Rubisch, Claudio Michaelis, Matthias Bethge, Felix A Wichmann, and Wieland Brendel. Imagenet-trained CNNs are biased towards texture; increasing shape bias improves accuracy and robustness. *arXiv preprint arXiv:1811.12231*, 2018. 2

[18] Amirata Ghorbani, Abubakar Abid, and James Zou. Interpretation of neural networks is fragile. *arXiv preprint arXiv:1710.10547*, 2017. 3

[19] Ian Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. In *International Conference on Learning Representations*, 2015. 6

[20] Guoliang Kang, Xuanyi Dong, Liang Zheng, and Yi Yang. Patchshuffle regularization. *arXiv preprint arXiv:1707.07103*, 2017. 2

[21] Yu Li, Peter Richtarik, Lizhong Ding, and Xin Gao. On the decision boundary of deep neural networks. *arXiv preprint arXiv:1808.05385*, 2018. 3

[22] 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. 4, 7

[23] Zachary C. Lipton. The mythos of model interpretability. *Queue*, 16(3):30:31–30:57, June 2018. 1, 3

[24] Stéphane Mallat. *A Wavelet Tour of Signal Processing*. Academic Press, 1998. 2, 4, 7

[25] Grégoire Montavon, Sebastian Lapuschkin, Alexander Binder, Wojciech Samek, and Klaus-Robert Müller. Explaining nonlinear classification decisions with deep taylor decomposition. *Pattern Recognition*, 65:211–222, 2017. 2, 3, 8

[26] Guido F Montufar, Razvan Pascanu, Kyunghyun Cho, and Yoshua Bengio. On the number of linear regions of deep neural networks. In *Advances in neural information processing systems*, pages 2924–2932, 2014. 6

[27] Konda Reddy Mopuri, Utsav Garg, and R Venkatesh Babu. CNN fixations: An unraveling approach to visualize the discriminative image regions. 2017. 3

[28] Pablo Navarrete Michelini, Hanwen Liu, and Dan Zhu. Multigrid backprojection super–resolution and deep filter visualization. In *Proceedings of the Thirty–Third AAAI Conference on Artificial Intelligence (AAAI 2019)*. AAAI, 2019, arXiv preprint arXiv:1809.09326. 2, 3, 4, 5, 7

[29] Chris Olah, Alexander Mordvintsev, and Ludwig Schubert. Feature visualization. *Distill*, 2017. https://distill.pub/2017/feature-visualization. 3

[30] Chris Olah, Arvind Satyanarayan, Ian Johnson, Shan Carter, Ludwig Schubert, Katherine Ye, and Alexander Mordvintsev. The building blocks of interpretability. *Distill*, 2018. https://distill.pub/2018/building-blocks. 3

[31] 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. 3, 7

[32] Terence Parr and Jeremy Howard. The matrix calculus you need for deep learning. *arXiv preprint arXiv:1802.01528*, 2018. 4

[33] John G. Proakis and Dimitris K. Manolakis. *Digital Signal Processing*. Prentice Hall international editions. Pearson Prentice Hall, 2007. 2, 4, 7

[34] Zhuwei Qin, Funxun Yu, Chenchen Liu, and Xiang Chen. How convolutional neural network see the world-a survey of convolutional neural network visualization methods. *arXiv preprint arXiv:1804.11191*, 2018. 3

[35] Marco T. Ribeiro, Sameer Singh, and Carlos Guestrin. Why should i trust you?: Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1135–1144. ACM, 2016. 3

[36] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958. 1, 3

[37] Matthias Rottmann, Pascal Colling, Thomas-Paul Hack, Fabian Hüger, Peter Schlicht, and Hanno Gottschalk. Prediction error meta classification in semantic segmentation: Detection via aggregated dispersion measures of softmax probabilities. *arXiv preprint arXiv:1811.00648*, 2018. 3

[38] 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. 5, 6

[39] Yousef Saad. *Numerical methods for large eigenvalue problems: revised edition*, volume 66. Siam, 2011. 5

[40] Avanti Shrikumar, Peyton Greenside, and Anshul Kundaje. Learning important features through propagating activation differences. In *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, pages 3145–3153, 2017. 3

[41] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014. 4, 5, 6

[42] Lawrence Sirovich and Michael Kirby. Low-dimensional procedure for the characterization of human faces. *J. Opt. Soc. Am. A*, 4(3):519–524, Mar 1987. 7

[43] Gilbert Strang. *Introduction to linear algebra*, volume 3. Wellesley-Cambridge Press Wellesley, MA, 1993. 2

[44] Gilbert Strang. The discrete cosine transform. *SIAM review*, 41(1):135–147, 1999. 2

[45] George Turin. An introduction to matched filters. *IRE transactions on Information theory*, 6(3):311–329, 1960. 5

[46] Aet Watanabe, T Mori, S Nagata, and K Hiwatashi. Spatial sine-wave responses of the human visual system. *Vision Research*, 8(9):1245–1263, 1968. 7

[47] Peng Xu, Bryan He, Christopher De Sa, Ioannis Mitliagkas, and Chris Re. Accelerated stochastic power iteration. In *International Conference on Artificial Intelligence and Statistics*, pages 58–67, 2018. 5

[48] Matthew D. Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *Computer Vision - ECCV 2014 - 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part I*, pages 818–833, 2014. 4, 5

[49] Quan-shi Zhang and Song-Chun Zhu. Visual interpretability for deep learning: a survey. *Frontiers of Information Technology & Electronic Engineering*, 19(1):27–39, 2018. 3

[50] Bolei Zhou, Yiyou Sun, David Bau, and Antonio Torralba. Interpretable basis decomposition for visual explanation. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 119–134, 2018. 3

[51] 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. 4, 8