Debugging the Internals of Convolutional Networks

Anonymous Author(s)
Affiliation
Address
email

Abstract

The filters learned by Convolutional Neural Networks (CNNs) and the feature maps these filters compute are sensitive to convolution arithmetic. Several architectural choices that dictate this arithmetic can result in feature-map artifacts. These artifacts can interfere with the downstream task and impact the accuracy and robustness. We provide a number of visual-debugging means to surface feature-map artifacts and to analyze how they emerge in CNNs. Our means help analyze the impact of these artifacts on the weights learned by CNNs. Guided by our analysis, model developers can make informed architectural choices that can verifiably mitigate harmful artifacts and improve the model's accuracy and its shift robustness.

1 Background and Motivation

The majority of work on visualizing convolutional neural networks (CNNs) has focused on understanding the features they learn [9, 18, 28]. Techniques such as feature saliency in a given input [3, 21, 23, 28], optimization-based feature visualization [27, 16, 18], and concept-based interpretation [8, 10] have improved our understanding of how CNNs work [20]. Furthermore, these techniques can expose certain limitations of CNNs such as classification decisions based on unreliable features and visual cues [11].

A number of techniques were proposed to explore various aspects of CNNs beyond interpreting the features and concepts they learn. For example, recent work has focused on visualizing the weights [19, 25] in order to analyze the role of individual filters and to expose patterns in a cohort thereof. Another piece of work has focused on branch specialization in CNNs [26]. Besides, several techniques have been proposed to analyze CNN predictions in order to expose patterns in erroneous ones [1, 4] and to subsequently shed light into the model's behavior.

In this work, we focus on exposing subtle effects in CNN innards caused by several aspects of convolution arithmetic [7]. These effects directly manifest in the feature maps computed within the CNN, in the form of artifacts. These effects can further manifest in the learned weights, in the form of asymmetries. Both issues have been partly discussed in a recent work [2]. Our contribution lies in providing generic visual means to inspect the feature maps and the learned weights for such artifacts and asymmetries. We demonstrate how debugging these effects help improve the accuracy and robustness of CNN-based models, besides deepening our understanding of their innards.

2 Visualizing Feature-Map Artifacts

A naive way to inspect feature maps inside CNNs for potential artifacts is to generate them for a number of images. However, this poses challenges regarding scalability and ability to spot the artifacts, as we explain next.

Submitted to 35th Conference on Neural Information Processing Systems (NeurIPS 2021). Do not distribute.

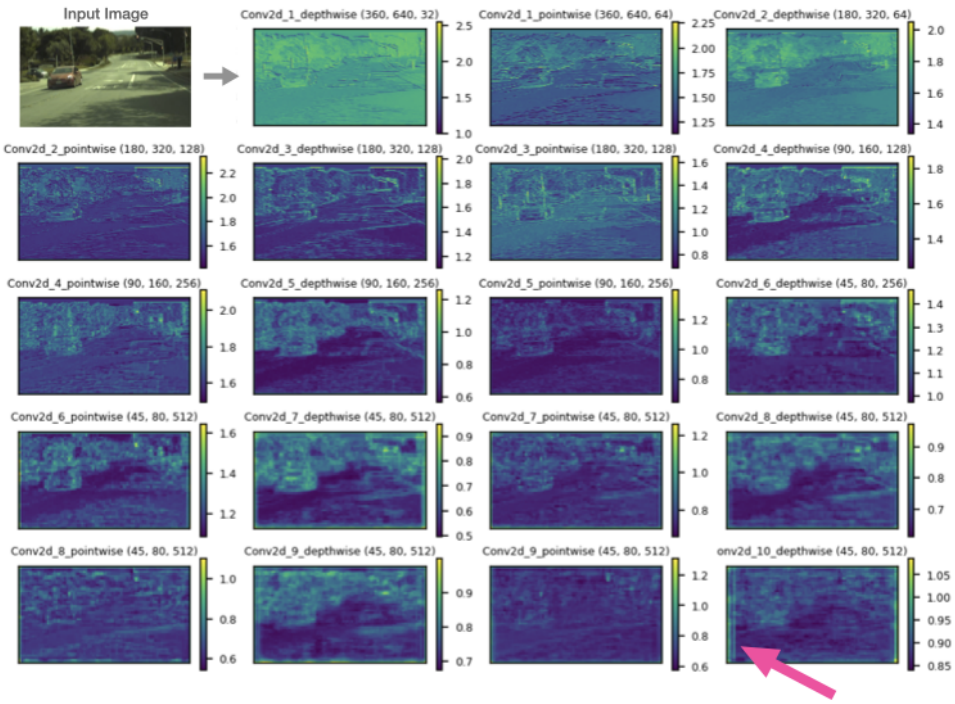

Figure 1: The per-layer mean feature map computed by an open-source SSD object detector [22] for a natural scene. The title indicates the spatial dimensions of the averaged feature maps, as well as their count. The pink arrow draws attention to line artifacts in the final map.

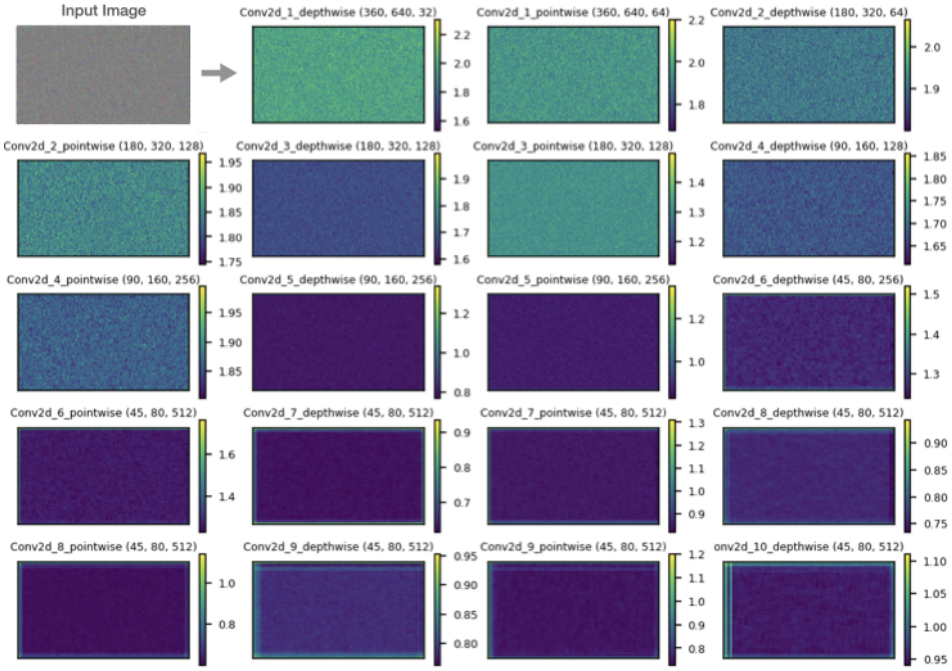

Figure 2: The same mean feature maps depicted in Figure 1, computed for a randomly generated input. This improves the visibility of the line artifacts in the final map.

2.1 Enabling Scalable Inspection

Each layer in a CNN typically computes hundreds of feature maps for every input. To enable scalable analysis, we aggregate the feature maps per layer to provide an overview. Figure 1 shows the mean feature map computed for a real scene in the CNN backbone of an SSD object detector [22].

Besides reducing the information overload, aggregation via averaging often helps surface systematic artifacts since it reinforces ones that recur in multiple maps while partially washing out benign areas in the maps. The final map in Figure 1, highlighted with a pink arrow, demonstrates an example of these artifacts. We will elaborate later on how convolution arithmetic can incur such artifacts.

2.2 Making the Artifacts Stand Out

Aggregating the feature maps of a layer only partially helps surface systematic artifacts in these maps. This is because the natural activation patterns can still interfere with these artifacts (Figure 1). To improve the visibility of the artifacts, we feed a baseline input into the CNN instead of natural images. We found the following baselines useful for that purpose:

Random inputs We generate k images uniformly at random and feed them to the CNN. Figure 2 depicts the mean feature map per layer, when using one random input ($k = 1$). These maps largely retain a random activation pattern, except for line artifacts that emerge around the boundaries at

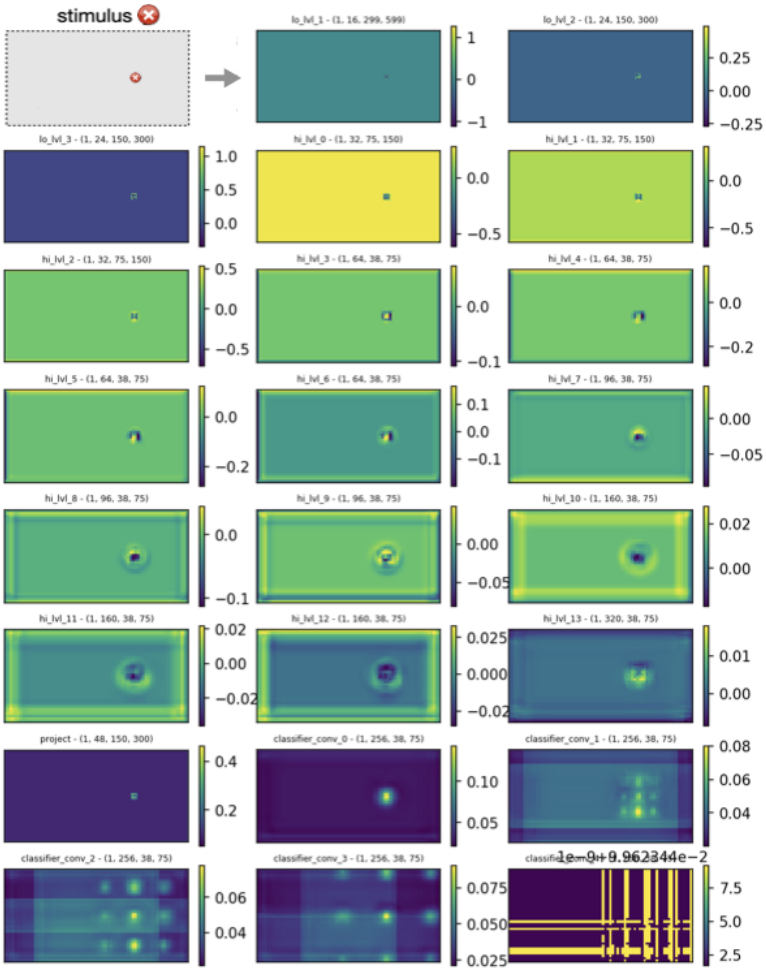

Figure 3: The per-layer mean feature maps computed by a DeepLab-V3 segmentation model for a simplified input that contains a stimulus. The model is trained on the CityScapes dataset [6]. The title indicates the spatial dimensions of the averaged feature maps, as well as their count.

deeper layers. This makes these artifacts stand out better than when feeding a natural input. When feeding multiple input samples ($k > 1$), we aggregate the feature maps per layer further over these samples which helps smoothening the mean maps.

Constant input With a constant input, the feature maps are largely constant. This enables analyzing subtle artifacts in these maps such as line, grid, and checker-board patterns (Figure 4). A zero-valued input (a gray image), a maximal input (a white image), or a minimal input (a black image) can be a suited choice, depending on the dataset and the model.

Simplified input The former two options eliminate selective and localized neuron activation that occurs in CNNs when feeding natural images, because they lack visual features that trigger such activation patterns. To analyze these patterns we can add a visual stimulus to a constant input and feed it to the CNN. Figure 3 demonstrates an example stimulus feed into a DeepLab-V3 semantic-segmentation model [5], trained on CityScapes [6]. This reveals artifacts that potentially impact how this stimulus is processed by the CNN. For example, in Figure 3 the stimulus leads to nine localized activation areas at layer`classifier_conv_1`, spaced out at an equal distance. Interestingly, this distance is equal to the dilation factor applied at this layer.

2.3 Drilling-down on Demand

Aggregation is useful to detect feature-map artifacts and to analyze their progression over successive layers. Besides, we enable users to drill-down into individual feature maps of a selected layer to

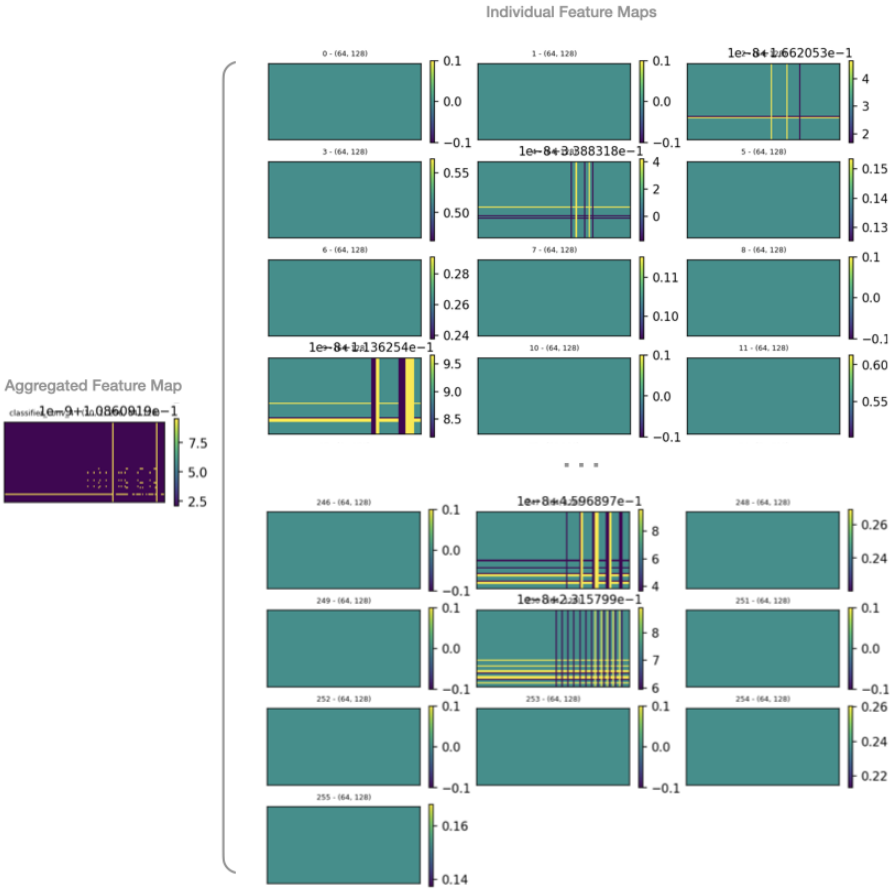

Figure 4: The individual feature maps at layer layer`classifier_conv_1` computed by the same model in Figure 3 for a zero-valued input. The title indicates the spatial dimensions of the averaged maps, as well as their count. This video demonstrates our interactive drill-down.

closely examine how these artifacts emerge. Figure 4 depicts a number of individual maps from layer `classifier_conv_1` of the aforementioned DeepLab-V3 model, generated by feeding a zero-valued input. While the majority of these maps are constant, several maps exhibit multiple line artifacts. These artifacts also appear when feeding a natural input, and can hence interfere with the extracted features and impact the segmentation quality.

2.4 What Causes the Artifacts?

The architectural choices that dictate the CNN arithmetic are behind the artifacts we demonstrate in previous sections. In particular, the padding mechanism, while fundamental, introduces an artificial boundary around the feature maps, especially under 0-padding. Over multiple convolutional layers, this boundary interacts with neighboring values and can gradually interfere with the activation patterns inside the feature maps beyond the boundary. This interference has been investigated in details in a recent work [2] that demonstrates its significant impact on small object detection. 0-padding was also shown to impact other tasks such as image classification [12] and semantic segmentation [12, 15]. The above studies propose effective mitigation via alternative padding schemes.

Another source of the artifacts is the downsampling mechanism used, such as pooling techniques and strided convolutions. This mechanism can amplify boundary effects and accelerate their interference with the rest of the feature maps. Moreover, the use of strides often incurs aliasing effects [29] and checkerboard artifacts [17]. Finally, downsampling can lead to skewness in the learned weights as we demonstrate in Section 3.

Besides the above-mentioned components of convolution arithmetic, dilation factors can also influence the emergence of feature-map artifacts and the patterns these artifacts take, as illustrated in Figure 3. Likewise, the input size also has an influence on these artifacts, since it is involved in the convolution arithmetic. Finally, non-arithmetic architectural choices can further amplify or dampen feature-map artifacts, such as normalization schemes and antialiasing [29].

3 Visualizing Asymmetries in the Learned Weights

Besides impacting model inference, feature-map artifacts can further impact the weights learned by the CNN during training. In particular, these weights might adapt to accommodate the presence of these artifacts during training. We aim to reveal such adaptation patterns in the learned weights. For this purpose we compute the mean $k \times k$ patch in the input of each convolution layer, averaged over the spatial positions, the channels, and the instances of the training set as Figure 5 illustrates. We also

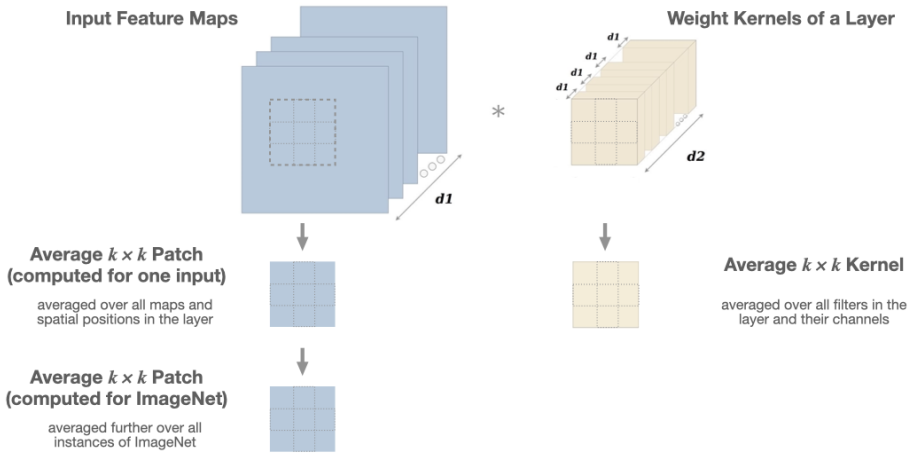

Figure 5: An illustration of the elements we use to analyze the impact of feature-map artifacts in a given layer on the weights it learns. Computing the mean $k \times k$ patch over the entirety of ImageNet is quite time-consuming. Fortunately, with a small k, a good approximation of this mean is possible with a few hundreds of samples.

Figure 6: Analyzing asymmetries in two ResNet-50 models trained on ImageNet using different input sizes. For each 3×3 convolutional layer, we depict the mean 3×3 patch in the layer's input and the mean kernel of its filters. Downsampling layers are highlighted in blue.

compute the mean $k \times k$ kernel in the weights of the corresponding layer. Figure 6 demonstrates our analysis on two ResNet-50 models trained on ImageNet images using two different sizes of the input images, 224×224 and 225×225. In each case, we depict the mean 3×3 patch in the padded input of each 3×3 convolutional layer in ResNet-50, besides the mean kernel of the layer weights.

Main Observations We notice that the majority of mean patches are symmetric about their centers. Surprisingly, all salient exceptions appear when the model is trained on 224×224 images. Moreover, patches the exhibit asymmetry are from the input to the three layers with $stride = 2$ (highlighted in blue) or the layers preceding them [1]. Interestingly, the mean kernels of these stride-convolution layers exhibit similar asymmetries, which is highly evident at `layer2.0.conv2` and `layer2.0.conv2`. Moreover, these asymmetries disappear when the model is trained with 225×225 images.

[1] One exception is the first depicted patch for `layer1.0.conv2`. This layer is preceded by a maxpool layer and a 2-strided convolutional layer (Figure 7), neither of which is depicted to focus Figure 6 on 3×3 filters.

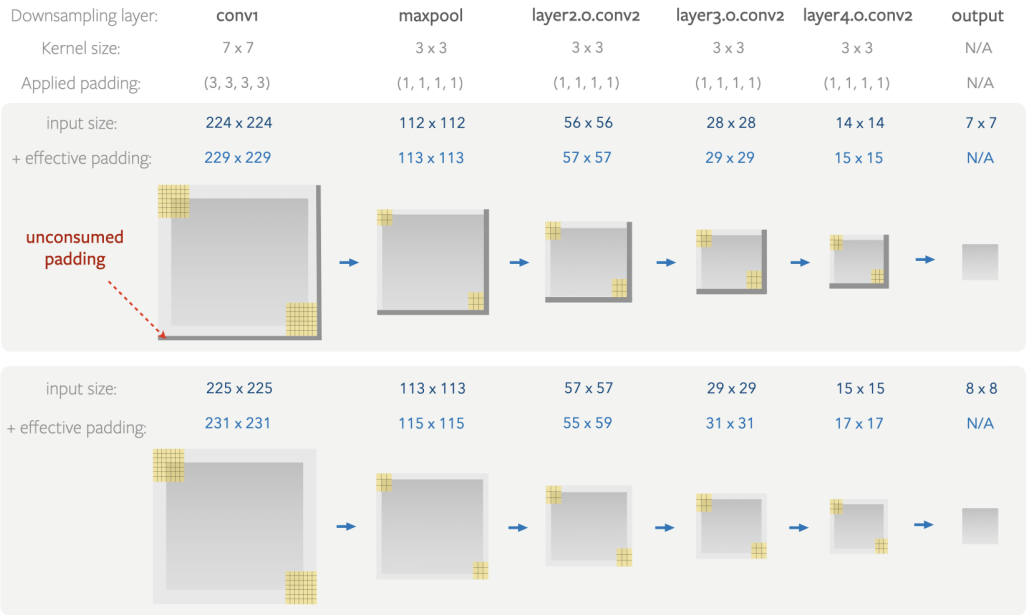

Figure 7: Illustrating the problem of uneven padding. The top row shows how an 224×224 input makes ResNet-50 ignore 1 pixel padding at the right and bottom side of every downsampling layer. In contrast, the bottom row shows how an 225×225 leaves no padding unconsumed.

Explaining the Asymmetries

To explain the above observations, we closely examine how downsampling layers in both models process their input, as illustrated in Figure 7. It is evident that these layers ignore parts of the padding applied when their input size is an even number. This happens at every downsampling layer when the CNN input size is 224×224. The unconsumed padding occurs only at the right and bottom sides of the feature maps. Since ResNet models fill the padding areas with zeros, failing to consume the padding at the right and bottom sides does change the statistics of the layer input. This explains why the mean 3×3 patches highlighted in Figure 6 exhibit asymmetries when the CNN input size is 224×224 input, thanks to the over-representation of zeros at the top and left borders. On the other hand, when this size is 225×225 the padding is fully consumed at all sides of the feature maps, making the mean patches highly symmetric.

Since the convolution involves multiplying 3×3 patches with corresponding 3×3 weight kernels, any asymmetries that govern these patches in the training set are likely to be reflected in the learned weights. This explains the strong asymmetries we observe in certain mean kernels when the CNN is training on 224×224 images [2]. This also explains the high symmetry of the mean kernels when training on 225×225 images.

In summary, the skewness we exposed collectively in the learned weights at certain layers is the CNN's way to adapt to the uneven application of padding to the input of these layers. This skewness is an artifact of CNN arithmetic: It does not capture inherent semantic information about the classification task since it disappears when uneven application of padding is eliminated. Ensuring even application of padding was shown to improve the accuracy and the translation invariance of the CNN on ImageNet [2]. This demonstrates the value of exposing potential artifacts of CNN arithmetic.

Limitations Exposing asymmetries in the learned weights becomes challenging with kernels larger than 3×3 in size, unless windowing is applied [24]. Moreover, the asymmetries might not necessarily be an artifact of CNN arithmetic, but rather reflect semantic properties of the dataset such as dominant vertical strokes in handwritten digits or letters. Analyzing the dataset features and examining the weight kernels under different hyperparameters can help determine possible reasons for asymmetry.

[2] One exception is the mean kernel at `layer2.2.conv2`. The mean patch in its input does not exhibit asymmetry. Interestingly, this asymmetry disappears when the model is trained on 225×225 images.

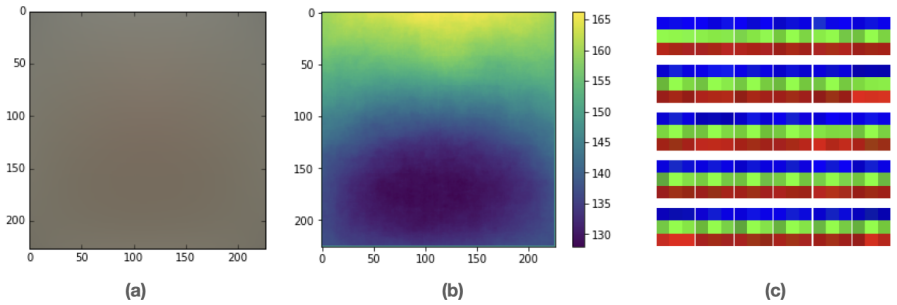

Figure 8: The impact of input statistics on banding effects in learned filters. (a) The mean image in the ImageNet dataset, depicted in the RGB space. (b) Visualizing the mean image after averaging over the channels. The color scale used better reveals the difference in brightness between the top and the bottom areas. (c) A visualization of a few filters from the final convolutional layer in a ResNet-50 model trained on ImageNet (reproduced from [19]). The visualization uses non-negative matrix factorization to reduce the depth to three channels which are mapped to the RGB space. The visible banding effect is induced by global-average pooling (GAP), as Petrov et al demonstrated [19]. The horizontal bands resemble the spatial differences in (b), confirming the role of the training dataset.

4 Lessons learned

We demonstrated how visual analysis of CNN internals can expose critical issues in CNNs that might otherwise go unnoticed. In fact, it has been known that certain architectural choices can introduce artifacts, however, CNNs were assumed able to implicitly handle them. We hope the following components of our analysis could inspire similar debugging and exposition techniques:

Baseline inputs Feeding a constant or near-constant input to a CNN helps simplify the internals it computes which, in turn, helps surface abnormalities in the internals. Feeding random inputs can also help in exposing systematic bias in these internals.

Aggregation and Ensemble Analysis Computing the mean feature map and the mean kernel in a layer is crucial to expose systematic spatial bias and asymmetries at that layer. Aggregation enables these patterns to emerge collectively, unlike the analysis of individual filters or feature maps.

Visual Representation Considering multiple visual representations of the model internals is helpful to iteratively reveal subtle patterns and variations that might influence the model behavior. Figure 8 demonstrates how a naive depiction of the mean image in ImageNet (Figure 8a) falls short of revealing spatial variations in this image, unlike a heatmap (Figure 8b). These variations help explain the banding effects in the learned weights, recently observed by Petrov et al [19] (Figure 8c), and confirm the role of the training data in the emergence of these effects.

Insights From Signal Processing Several phenomena related to CNNs have already been studied in signal and image processing. For example, zero padding has been known to cause image artifacts [13], with several alternatives proposed in the literature to mitigate them [14]. Likewise, insights from sampling theory have helped reducing aliasing effects in CNNs [29], and windowing techniques were shown to reduce spectral leakage and the artifacts it induces in CNN internals [24].

5 Conclusion

Convolution arithmetic can be a source of artifacts that impact the innards of CNNs. These artifacts manifest both in the feature maps and in the weight kernels. We presented visual means to expose these artifacts, to analyze why and how they emerge, and to understand their impact on the task. Our open-source means leverage the power of visualization along with deliberate inputs and aggregation schemes. We exemplified several insights enabled by our analysis and demonstrated how they help model developers make informed decisions to mitigate the aforementioned artifacts and biases, and as a result improve the accuracy and robustness of their models. Future work can dive deeper

into how the effects we expose impact specific CNN tasks, and whether similar effects appear in transformer-based models and hybrid CNN and transformer models.

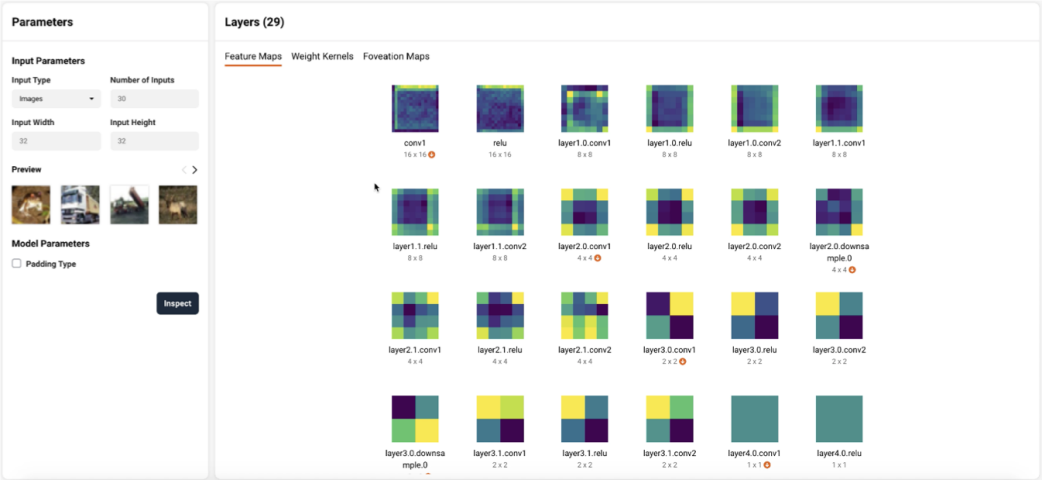

Figure 9: A screenshot of our interactive visual-debugging tool showing an overview of the feature-map artifacts in a ResNet-18 model. The small red arrows aim to mark down-sampling layers. This anonymized video demonstrates how users can generate these feature maps for different baseline inputs and under different hyper-parameter settings. The video also demonstrates how users can drill-down into the aggregated maps and analyze the impact of the artifacts on the weight kernels. Our tool will be made available as an open-source software.

References

[1] Alsallakh, B., Jourabloo, A., Ye, M., Liu, X., and Ren, L. (2017). Do convolutional neural networks learn class hierarchy? *IEEE transactions on visualization and computer graphics*, 24(1):152–162.

[2] Anonymized (2021). Mind the pad – cnns can develop blind spots. In *International Conference on Learning Representations (ICLR)*.

[3] Bach, S., Binder, A., Montavon, G., Klauschen, F., Müller, K.-R., and Samek, W. (2015). On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation. *PloS One*, 10(7).

[4] Boggust, A., Hoover, B., Satyanarayan, A., and Strobelt, H. (2021). Shared interest: Large-scale visual analysis of model behavior by measuring human-ai alignment. *arXiv preprint arXiv:2107.09234*.

[5] Chen, L.-C., Papandreou, G., Schroff, F., and Adam, H. (2017). Rethinking atrous convolution for semantic image segmentation. *arXiv preprint arXiv:1706.05587*.

[6] Cordts, M., Omran, M., Ramos, S., Rehfeld, T., Enzweiler, M., Benenson, R., Franke, U., Roth, S., and Schiele, B. (2016). The cityscapes dataset for semantic urban scene understanding. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3213–3223.

[7] Dumoulin, V. and Visin, F. (2018). A guide to convolution arithmetic for deep learning. *arXiv preprint arXiv:1603.07285*.

[8] Ghorbani, A., Wexler, J., Zou, J. Y., and Kim, B. (2019). Towards automatic concept-based explanations. *Advances in Neural Information Processing Systems*, 32:9277–9286.

[9] Grün, F., Rupprecht, C., Navab, N., and Tombari, F. (2016). A taxonomy and library for visualizing learned features in convolutional neural networks. In *ICML Workshop on Visualization for Deep Learning*, page 8.

[10] Kim, B., Wattenberg, M., Gilmer, J., Cai, C., Wexler, J., Viegas, F., et al. (2018). Interpretability beyond feature attribution: Quantitative testing with concept activation vectors (tcav). In *International conference on machine learning*, pages 2668–2677. PMLR.

[11] Lapuschkin, S., Wäldchen, S., Binder, A., Montavon, G., Samek, W., and Müller, K.-R. (2019). Unmasking clever hans predictors and assessing what machines really learn. *Nature communications*, 10(1):1–8.

[12] Liu, G., Shih, K. J., Wang, T.-C., Reda, F. A., Sapra, K., Yu, Z., Tao, A., and Catanzaro, B. (2018). Partial convolution based padding. In *arXiv preprint arXiv:1811.11718*.

[13] Liu, R. and Jia, J. (2008). Reducing boundary artifacts in image deconvolution. In *IEEE International Conference on Image Processing (ICIP)*, pages 505–508.

[14] Lou, S., Jiang, X., and Scott, P. J. (2011). Fast algorithm for morphological filters. *Journal of Physics: Conference Series*, 311(1):012001.

[15] Murase, R., Suganuma, M., and Okatani, T. (2020). How can cnns use image position for segmentation? *arXiv preprint arXiv:2005.03463*.

[16] Nguyen, A., Dosovitskiy, A., Yosinski, J., Brox, T., and Clune, J. (2016). Synthesizing the preferred inputs for neurons in neural networks via deep generator networks. In Lee, D., Sugiyama, M., Luxburg, U., Guyon, I., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc.

[17] Odena, A., Dumoulin, V., and Olah, C. (2016). Deconvolution and checkerboard artifacts. *Distill*, 1(10):e3.

[18] Olah, C., Mordvintsev, A., and Schubert, L. (2017). Feature visualization. *Distill*, 2(11):e7.

[19] Petrov, M., Voss, C., Schubert, L., Cammarata, N., Goh, G., and Olah, C. (2021). Weight banding. *Distill*, 6(4):e00024–009.

[20] Samek, W., Montavon, G., Vedaldi, A., Hansen, L. K., and Müller, K.-R. (2019). *Explainable AI: interpreting, explaining and visualizing deep learning*, volume 11700. Springer Nature.

[21] Selvaraju, R. R., Cogswell, M., Das, A., Vedantam, R., Parikh, D., and Batra, D. (2017). Grad-cam: Visual explanations from deep networks via gradient-based localization. In *Proceedings of the IEEE international conference on computer vision*, pages 618–626.

[22] Shalnov, E. (2019). BSTLD-demo: A sample project to train and evaluate model on BSTLD. https://github.com/e-sha/BSTLD_demo.

[23] Sundararajan, M., Taly, A., and Yan, Q. (2016). Gradients of counterfactuals. *arXiv preprint arXiv:1611.02639*.

[24] Tomen, N. and van Gemert, J. (2021). Spectral leakage and rethinking the kernel size in cnns. *arXiv preprint arXiv:2101.10143*.

[25] Voss, C., Cammarata, N., Goh, G., Petrov, M., Schubert, L., Egan, B., Lim, S. K., and Olah, C. (2021a). Visualizing weights. *Distill*, 6(2):e00024–007.

[26] Voss, C., Goh, G., Cammarata, N., Petrov, M., Schubert, L., and Olah, C. (2021b). Branch specialization. *Distill*, 6(4):e00024–008.

[27] Yosinski, J., Clune, J., Fuchs, T., and Lipson, H. (2015). Understanding neural networks through deep visualization. In *In ICML Workshop on Deep Learning*.

[28] Zeiler, M. D. and Fergus, R. (2014). Visualizing and understanding convolutional networks. In *European conference on Computer Vision (ECCV)*, pages 818–833.

[29] Zhang, R. (2019). Making convolutional networks shift-invariant again. In *International Conference on Machine Learning (ICML)*.