Do We Need to Learn Spatial Mixing?

George Cazenavette
Carnegie Mellon University

Joel Julin
The University of Pittsburgh

Simon Lucey
The University of Adelaide

Abstract

Until quite recently, the backbone of nearly every state-of-the-art computer vision model has been 2D convolution. At its core, a 2D convolution mixes both in the spatial dimensions and the channel dimension. Many recent computer vision architectures consist of sequences of isotropic blocks that separate out the spatial and channel-mixing components. In this paper, we explore the question: "Do we need to learn spatial mixing?" On both classical (ResNet) and cutting edge (ConvMixer) models, we find that we can reach the same performance by learning only the channel mixing portions of the network. In some cases, this drastically reduces training time and improves adversarial robustness. We hope that this work opens up new insights into spatial mixing's role in the success of modern networks.

1. Introduction

For the better part of the last two decades, cascades of learned convolutions have formed the backbone of nearly every innovation in the field of computer vision and pattern recognition. From distinguishing ten digits with LeNet [17] to one thousand classes with AlexNet [16], from going very deep with VGG [22] to even deeper with InceptionNet [23], the learned 2D convolution has served as the workhorse that ushered in the new era of visual learning. Convolutional neural networks (CNNs) learned to exploit the correlations across channels and between spatially close pixels to solve a plethora of tasks.

Recently, isotropic networks (those in which the size of the representation stays fixed) such as Vision Transformer [8], Image GPT [5], MLP-Mixer [24], and ConvMixer [2] have been grabbing the field's attention. These isotropic models consist of repeated blocks wherein each block consists of a spatial mixing operation (self-attention [5, 8], spatial MLP [24], or depthwise convolution [2]) followed by one or more *strictly* channel-mixing layers (1×1 or "pointwise" convolutions). A strictly spatial mixing operation is defined here as any operation that is applied independently across each channel. Equally, a strictly channel-mixing layer is applied independently across spatial pixel coordinates. The fact that all these recent state-of-the-art isotropic architectures spend a significant portion of their computation budget solely on channel-mixing parameters hints toward the possibility that channel mixing may have significantly more relative importance than previously appreciated.

It has been understood for some time that impressive performance can be obtained from networks whose weights are not all completely learned [21]. Before the success of AlexNet [16] in training deep end-to-end networks, the use of random weights in early convolutional layers played an important role in training deeper CNNs. The approach was attractive as it promised faster training times, better generalisation and the ability to learn deeper networks. Saxe et al. [21] even offered strong theoretical motivations for why random filter weights in CNNs would offer good performance in terms of: (i) frequency selection, and (ii) translation invariance. One of the most notable works with respect to using random weights within networks can be found in the Extreme Learning Machines (ELMs) of Huang et al [13]. In its simplest form an ELM takes a proposed network architecture and randomly initializes all hidden weights leaving only the final layer to learn. ELMs are advantageous as they allow for extremely deep networks and rapid train times (as only the final layer needs to be learned). It is widely understood, however, that current training strategies have a significant performance gap in relation to their current end-to-end learned counterparts.

In this paper, we revisit the idea of using random weights within a CNN. Our work differs to previous works on random weights in that we choose to *only* not learn spatial mixing parameters. For both mainstream (ResNet) and exotic (ConvMixer) architectures, we uncover a trend wherein the learned channel-mixing portion of a network tends to be the most **performance-critical**, **computationally efficient**, and **adversarially robust**. In general, we show neural networks that *only learn channel mixing* can reach nearly the same performance as their fully-learned counterparts while also being computationally cheaper and more resistant to adversarial perturbations. We hope these insights to be of value to the computer vision community and that they will shed light on which parts of a neural network are truly necessary to learn.

2. Related Work

Isotropic Architectures First popularized by the transformer [25], isotropic architectures have more recently made their way into the computer vision world. ImageGPT [5] (based on the GPT language model [5]) was able to model sequences of pixels and generate new images in raster fashion. Designed for discriminative tasks, the Vision Transformer [8] used transformer blocks to achieve state-of-the-art image classification results, but only after extensive pre-training. The MLP-Mixer [24] then built off Vision Transformer's success, but used a simple spatial-mixing MLP instead of the expensive self-attention module. While the MLP-Mixer still required extensive pre-training, the newest addition to the line of isotropic architectures, the ConvMixer [2], replaces the self-attention or spatial-mixing MLP with a simple depthwise filter bank and achieves comparable results *without* any pre-training. The main thing all these models have in common is their *isotropic* structure. Here, "isotropic" refers to repeated blocks of operations in which the latent representation does not change in shape. Furthermore, in all of these models, the second half of each isotropic block is made up of one or more *strictly-channel-mixing* layers.

Separable Convolutions Instead of mixing across channel and spatial dimensions simultaneously, the separable convolution factorizes the operation into a depthwise and pointwise convolution. The depthwise convolution applies a disjoint set of depth-1 filters to each channel of the input independently while the pointwise convolution is simply a 1×1 convolution (or linear projection) on the pixels of this intermediate representation. Separable convolutions are used extensively in many state-of-the-art architectures [6, 12, 26] and have built-in implementations in deep learning libraries [1].

3. Background and Setup

Before the rise of self-attention [25] and transformer-based architectures like Vision Transformer [8] and ImageGPT [5], the standard 2D convolution formed the backbone of nearly every state-of-the art image classification model. Starting with the original CNN, LeNet [17], and moving forward through history with the likes of AlexNet [16], VGG [22], InceptionNets [23], and ResNets [10], the classical signal processing operation coupled with a learnable filter bank gave neural networks the power to generalize to unseen data in ways not possible with fully-connected networks.

3.1. Spectral Coverage

Part of the efficacy of a spatial convolution can be attributed to the size of the *spectral envelope* of its filter bank,

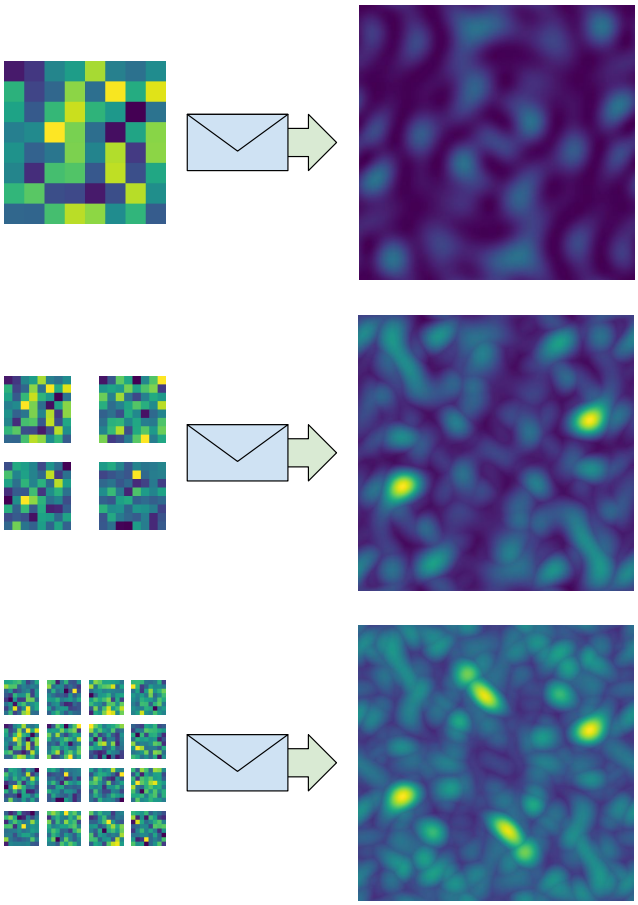


Figure 1. The spectral envelope of a bank of random filters rapidly grows as we add more uncorrelated filters to the set, allowing for quality feature extraction from natural images.

or how much *spectral coverage* it has. By spectral coverage, we mean the number of signal frequencies covered by the filter bank. By using multiple filters in tandem, the region of the envelope grows to be the union of each filter's individual coverage.

Previous work [13, 21] has shown that random filters make adequate visual feature extractors. Furthermore, motivated by decades of work in signal processing, it has been known that random filters have good spectral coverage. We hypothesize that these phenomena are related, and that the success of random filters is due in part to the size of their induced spectral envelope. As seen in Figure 1, increasing the number of filters rapidly increases the bank's coverage in frequency space.[1]

However, due to the nature of the convolution operation, to have good spectral coverage, the filters must also be uncorrelated from each other's *translations*. As such, using a

[1]We visualize the spectral envelope by taking the max over the magnitude of the 2D fast Fourier transform (FFT) of each filter padded to be 512×512. In our visualization the zero frequency component has been shifted to the center of the image.

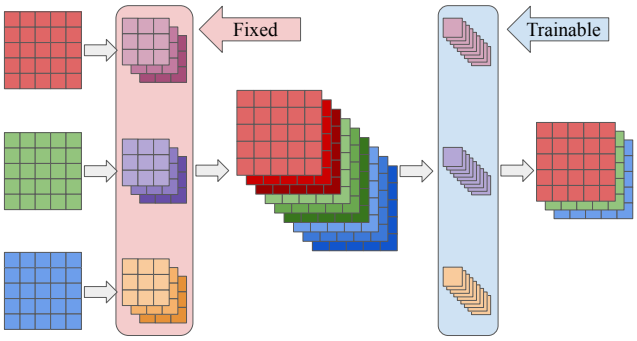


Figure 2. Separable convolutions that only learn channel mixing perform nearly as well as fully-learned separable convolutions. This type of convolution is denoted "Sep-Chans" in our results.

simple blur or even the same random initialization for every filter in the bank will result in *exceptionally poor* spectral coverage. A notable downside to fully random filters is that they unnecessarily cover high frequencies which has the consequence of leaving them vulnerable to *adversarial attacks* (targeted perturbations meant to fool a network). In Section 4.3, we will go even further and show benefits of forcing random filters to to focus on lower frequencies with respect to these adversarial attacks.

3.2. The 2D Convolution

As we begin our investigation into the importance of channel mixing, we focus on the simple yet revolutionary ResNet [10] architecture. The original ResNet architecture is composed of blocks of 2D convolution operations. Given the number of in and out channels (c_{in}, c_{out}) and the size of the square kernel (k), we can represent the number of trainable parameters in the standard 2D convolution (p_{conv}) as

$$p_{conv} = c_{in} \cdot c_{out} \cdot k^2 \tag{1}$$

With the vanilla Conv2D, it is hard to analyze the respective importance of spatial and channel mixing since the two are entangled in a single linear operation.

To disentangle the spatial and channel mixing [6, 12, 26], we separate the standard 2D convolution operation into a paired spatial-mixing (depthwise) and channel-mixing (1×1 or pointwise) convolution. In a separable convolution, each filter of the depthwise convolution operates on just a single channel of the input. The pointwise convolution is simply a 1×1 convolution, or a linear projection of each pixel.

After separating the standard 2D convolution into a depthwise and pointwise convolution, we can then introduce a *depth multiplier* to increase the number of filters per input channel. Naturally, this also increases the number of channels in the pointwise filters since the intermediate representation will now have more channels. Given the number of in and out channels (c_{in}, c_{out}), the size of the square depthwise kernel (k), and the depth multiplier d, we can represent the number of trainable parameters in the separable 2D convolution (p_{sep}) as

$$\begin{aligned} p_{sep} &= p_{depth} + p_{point} \\ &= c_{in} \cdot k^2 \cdot d + c_{in} \cdot d \cdot c_{out} \end{aligned} \tag{2}$$

By exploiting the inherent dependencies between nearby pixels, the standard 2D convolution—and its separable cousin—offer vision systems a cheap, yet effective way of extracting information from images.

3.3. The ConvMixer

As the latest (at the time of writing) in an ever-growing line of isotropic vision models [5, 8, 24], the ConvMixer [2] architecture adapts the (at this point) classical method of convolutions to the equi-sized representation (isotropic) paradigm of transformer models. Perhaps more importantly for our analysis, it serves as a state-of-the-art convolutional model in which *the convolutions are already separable*. In other words, the ConvMixer consists of strictly depthwise and pointwise convolutions. This allows us to examine the computational benefits of learning only channel mixing without the added overhead of converting the standard convolutions into separable ones.

Specifically, the ConvMixer [2] architecture consists of depthwise convolutions (wrapped in a residual connection) and pointwise convolutions. Specifically, a depth-n ConvMixer consists of a one-layer patch-projection stem, n depthwise-pointwise blocks, a global average pool, and a linear classifier. All throughout, the representation maintains the size to which it is projected by the stem (hence the term *isotropic*).

3.4. Is Channel Mixing All You Need?

Unlike the spatial-mixing component of separable convolution, the channel-mixing has no such constraints on its coherence. The atoms of the linear projection need only be *directly* uncorrelated with each other; there is no translationary correlation to worry about.

By isolating the spatial and channel-mixing parts of the standard convolution into separable components, we can analyze the contribution of channel mixing alone to the success of convolutional neural networks. To do this, we now introduce a form of the separable convolution (Figure 2) wherein we leave the depthwise (spatial-mixing) filters frozen at their initialized values and only learn the pointwise (channel-mixing) filters during training. **For simplicity, we will refer to this as a "static" convolution.** Given the number of in and out channels (c_{in}, c_{out}) and the depth

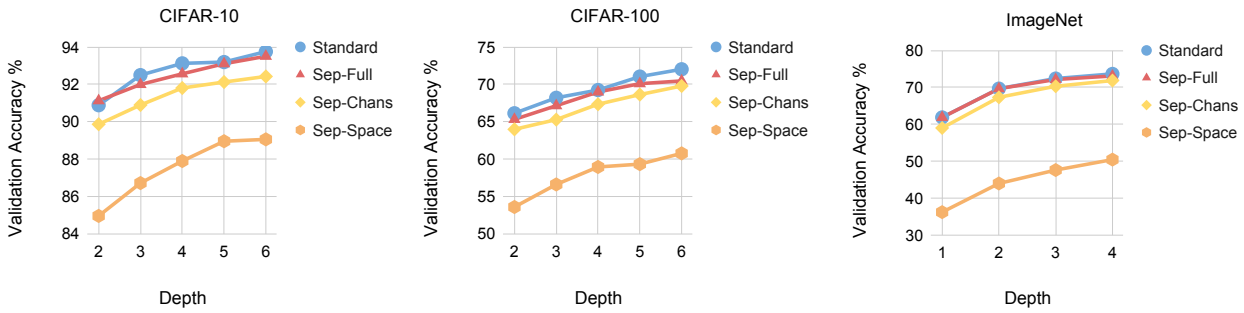


Figure 3. While the fully learned ResNets (Standard, Sep-Full) consistently outperform all others, we see that the models that only learn channel mixing (Sep-Chans) remain competitive. Conversely, the models that only learn spatial mixing (Sep-Space) lag very far behind the others.

multiplier d , we can represent the number of trainable parameters in the static 2D convolution (p_{static}) as

$$p_{static} = p_{point} = c_{in} \cdot d \cdot c_{out} \tag{3}$$

Note that the number of trainable parameters does not depend on the size of the kernel (k) since only the pointwise parameters are learned. Furthermore, we can ensure the number of trainable parameters in the static convolution is equal to that of the corresponding standard convolution by setting $d = k^2$:

$$\begin{aligned} & d = k^2 \\ \implies & c_{in} \cdot c_{out} \cdot k^2 = c_{in} \cdot d \cdot c_{out} \\ \implies & p_{conv} = p_{static} \end{aligned} \tag{4}$$

For ResNet architectures, the standard convolution kernel is of shape 3×3, so we would set $d = 3^2 = 9$ to use the same number of parameters in our static convolution (and $d = 3$ to use one third of the parameters).

4. Experiments

After motivating with some relevant background, we now begin our study into the capabilities of networks that only learn channel mixing. Section 4.1 applies our hypothesis to classical ResNet [10] architectures. Then in Section 4.2, we move on to the naturally separable ConvMixer [2] architecture to illustrate further computational advantage. Lastly, in 4.3, we explore a practical application in the form of architectural adversarial robustness. Our code will be made public upon publication.

4.1. ResNet Experiments

For these experiments, we train all models under identical conditions: the same number of epochs, same batch size, same learning rate, same decay schedule, etc. Any and all differences (or similarities) in performance are due entirely to the intrinsic properties of the architectures themselves.

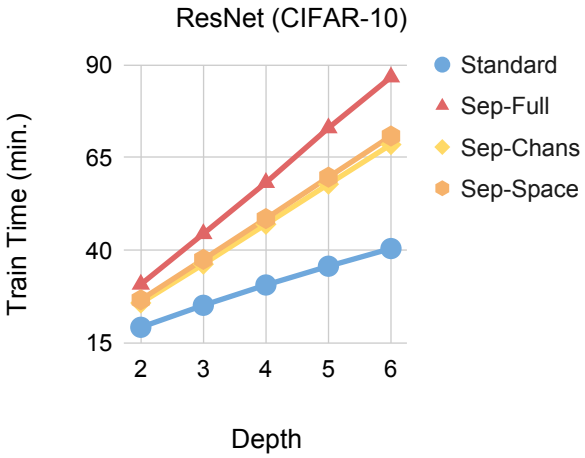


Figure 4. We see a large increase in speed when using a "static" separable network versus a learned separable one, but they are still held back by the overhead of converting the standard convolutions to separable ones.

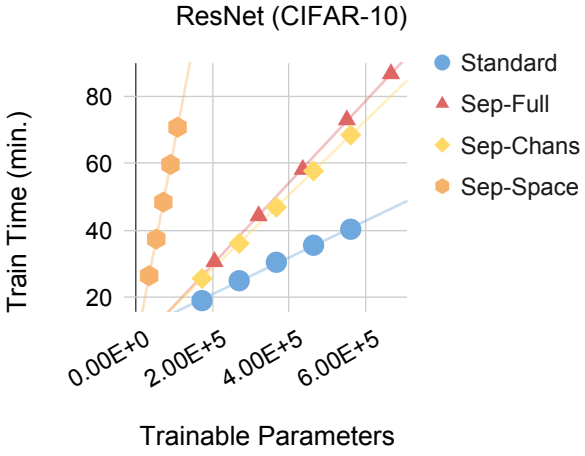


Figure 5. Despite having far-fewer trainable parameters, the spatial-mixing-only networks still take just as long to train as the channel-mixing-only networks. Spatial mixing parameters seem to have a much higher computational cost than channel mixing.

4.1.1 Model Architectures

For our first set of experiments, we employ the ResNet [10] architecture with the identity mapping modifications introduced in [11]. A depth-n CIFAR [15] ResNet contains a 1-layer stem, $2n$ layers for each of the 3 blocks, and a linear classifier, for a total of $6n + 2$ layers. Similarly an ImageNet [7] ResNet contains a 1-layer stem, $2n$ layers for each of the 4 blocks, and a linear classifier, for a total of $8n + 2$ layers. The "depth" in our plots represents this n. Unless otherwise specified, all separable ResNets have a depth multiplier of 9.

4.1.2 Classification Gap

Since the penultimate layer of ResNet is a global average pool, the linear classifier at the end is another strictly channel-mixing operation. Thus, our ResNets that use the static convolutions (Sep-Chans) **learn only channel mixing** throughout the entire network. Any spatial mixing is done using depthwise filters fixed at their random initializations.

When we look at the performance of ResNets using the standard convolution versus those learning *only* channel mixing (Sep-Chans), we make a somewhat shocking discovery: given the same number of trainable parameters, the gap in classification accuracy is barely there at all (Figure 3). One might question this observation, noting that CIFAR [15] is a simple dataset, and the performance may just be saturated. Yet, the trend also holds for ImageNet [7], a *much* larger and more complex dataset. Furthermore, we also see that networks only learning spatial mixing (Sep-Space) consistently perform *significantly* worse than those learning only channel mixing.

One might note that the number of spatial mixing parameters in a separable convolution, learned or not, is inherently smaller than that of channel mixing parameters. This is true; however, the same can also be said of the standard convolution. In a typical network, the channel mixing dimensions that contribute to the weight size ($c_{in} \cdot c_{out}$) are typically much larger than the spatial mixing dimensions ($k \cdot k$).

Ultimately, the shown fact that networks learning only channel mixing achieve competitive results to those that are fully learned comes as quite the surprise and calls into question the significance of learning the spatial and channel mixing operations in general.

4.1.3 Computational Considerations

A forward pass and back-propagation through an unlearned layer should be significantly faster than learnable one. We see this clearly in Figure 4: the networks with fully-learned separable convolutions (Sep-Full) take significantly longer to train than those with frozen spatial or channel-mixing layers (Sep-Space, Sep-Chans).

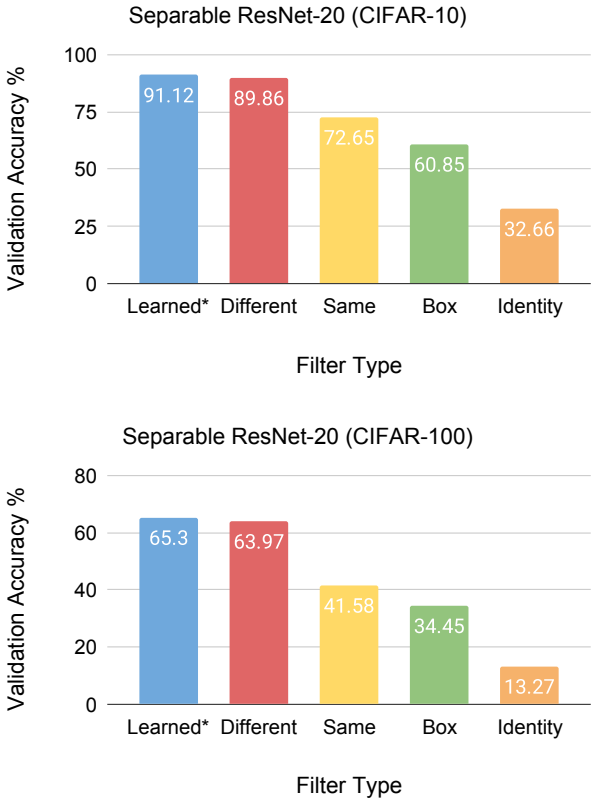
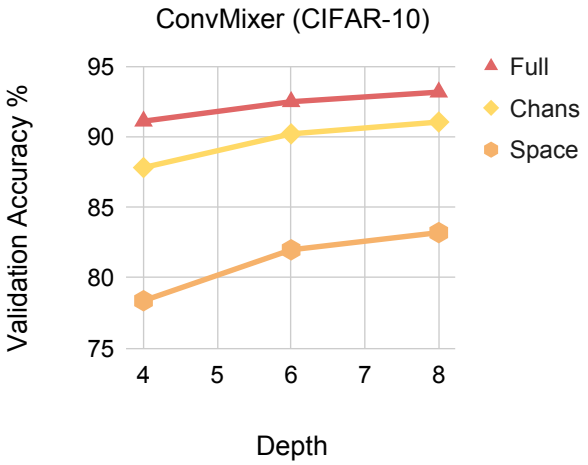


Figure 6. While *randomly* initialized filters can provide competitive results, the same is not true for *any* arbitrary, fixed filter.

However, the separable convolutions that learn only channel mixing (Sep-Chans) are still slower than the fully-learned standard (not separable) convolutions. Although *these* separable convolutions do not learn any spatial mixing, converting from a standard to a separable convolution still requires doubling the amount of total convolution operations (learned or not) in the network, significantly increasing the overhead.

In Figure 5, we see that the networks only learning spatial mixing (Sep-Space) have a much higher ratio of compute time per trainiable parameter than the fully learned separable networks (Sep-Chans). At first, this might just seem due to the spatial-only networks simply having fewer trainable parameters than the fully learned ones. However, we also see that the networks only learning channel mixing (Sep-Chans), which *also* have fewer trainable parameters than the fully trainable ones, have a *lower* train time to trainable parameter ratio than the fully learned separable convolutions. With these two observations in mind, we can infer that learnable spatial-mixing parameters have a higher associated computational cost than learnable channel-mixing parameters.



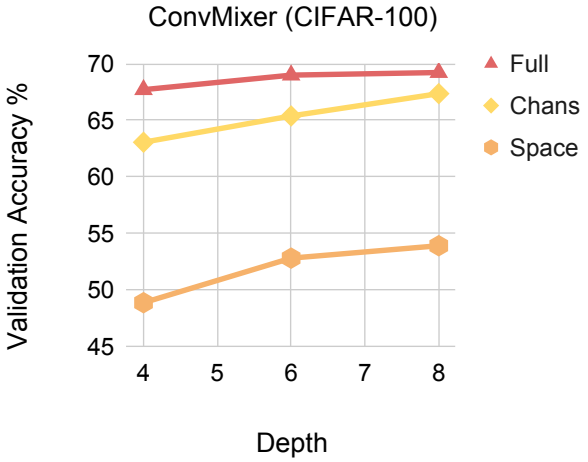


Figure 7. With the ConvMixer architecture, we can better analyze the direct contributions of spatial and channel mixing without altering the original model. We again see networks that only learn channel mixing (Chans) remaining competitive with their fully-learned counterparts (Full).

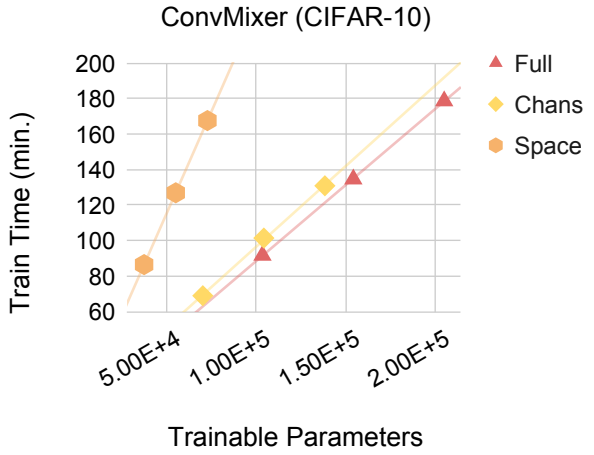


Figure 8. Despite having *much* fewer trainable parameters than the channel-learning type, the spatial-learning ConvMixers have a significantly longer train time.

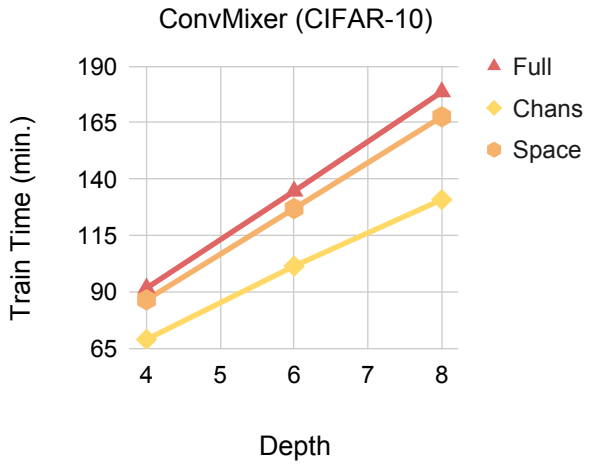


Figure 9. We only see a minimal time save by not learning the channel mixing, but there is significant speedup when not learning the spatial mixing. A much larger portion of the total train time of the fully-learnable version seems to be taken up by the spatial mixing parameters than the channel mixing counterparts.

4.1.4 Static Filter Structure

While we have shown that learning spatial mixing is not necessary to achieve competitive performance without changes to the model's architecture, the static filters cannot be *completely* arbitrary; they must still hold some basic properties to adequately transform the input signal for the reasons discussed in Section 3.1. As we speculated in our section on spectral coverage, the fixed filters with the largest spectral envelope have the best validation accuracy, and performance quickly degrades with the level of spectral coverage.

Looking at Figure 6, we see the performance of the fully-learned separable convolutions as the leftmost (blue) column. The next column (red) represents the separable convolution that only learns channel mixing. We again note that the randomly initialized spatial mixing performs only slightly worse than the fully-learned version. We label this column "Different" to contrast with the next (yellow) column, "Same." For this experiment, we apply the *same* set of 9 randomly initialized kernels to *every input channel*. The large performance drop here shows us that highly-correlated filters do not make good feature extractors since they have a very small spectral envelope. The next drop-off to the box filter (green) occurs because box filters are *perfectly* translationally correlated with each other. Lastly, the catastrophic failure of the identity filter (orange) tells us that although we might not need to *learn* it, spatial mixing is still an integral part of discriminative networks.

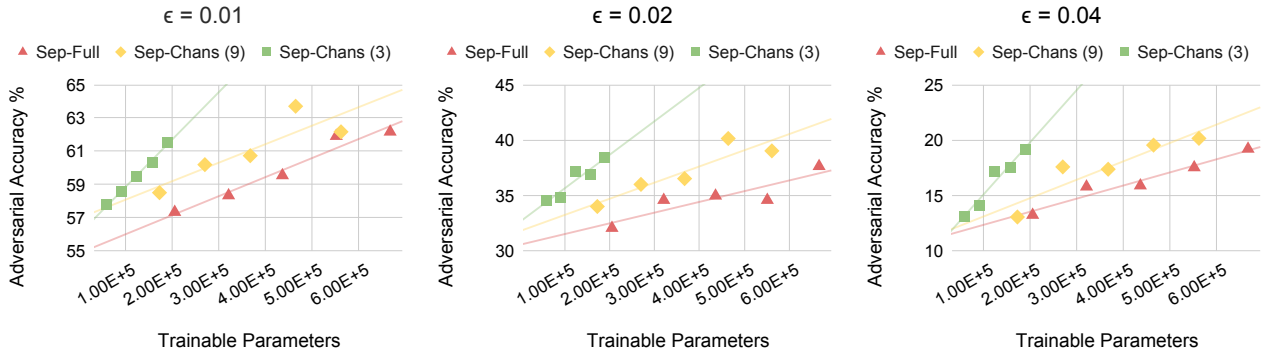


Figure 10. Adversarial accuracy in separable ResNets (CIFAR-10). The networks with static spatial mixing (Sep-Chans) are clearly more robust to adversarial perturbations than their learned counterparts (Sep-Full) in figure. Static networks with a lower depth multiplier (3 versus 9) seem to be even more robust per trainable parameter. (ϵ is the magnitude of the adversarial perturbations.)

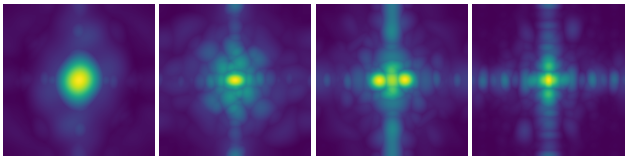


Figure 11. Spectral envelopes of learned ConvMixer filters in Figure 12. Naturally-learned filters have dense coverage of the lower frequencies without unnecessarily capturing the high frequencies.

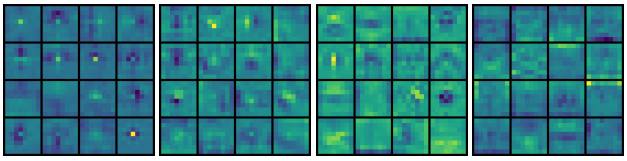


Figure 12. Learned depthwise filters from a ConvMixer. Learned filters seem to follow different distributions depending on layer.

4.2. ConvMixer Experiments

We use the code provided along with the ConvMixer paper and a model with 128 channels and a patch-size of 1 across various depths. Here, we discard the "Sep" in figures since the ConvMixer is already naturally separable.

4.2.1 Naturally Separable Performance

Now that we are dealing with a naturally separable architecture, all comparisons can be made *directly* with the original model. In Figure 7, we again see that if we learn only channel mixing, we still achieve results competitive with the fully-learned architecture.

As we saw with ResNet (Figure 5), the spatial-learning ConvMixers have a much higher ratio of train time to trainable parameter count than the channel-learning models, as seen in Figure 8. This observation is again validated by the fact that the channel-learning ConvMixers have a significantly shorter train time than the spatial-learning ConvMixers despite having higher trainable parameter counts.

4.2.2 Filter Observations

After years of networks using mostly 3×3 filters, the ConvMixer [2] takes a step back to larger filters that can be more easily understood by the human eye. In Figure 12, we see the learned filters of a depth-4 ConvMixer. The learned filters are clearly more structured than their random counterparts, and the filters of each layer seem to follow a different distribution.

4.3. Adversarial Robustness

Another practical application of our discovery lies in the adversarial setting. Adversarial examples are those specifically curated to fool a neural network into making a mistake at inference time by adding targeted noise to the sample [9]. Typically, the adversarial perturbations are small enough in magnitude that the semantic meaning of the example does not change and that a human observer cannot even notice them.

4.3.1 The Fast Gradient Sign Method

First, we will quickly introduce one of the first and most simple adversarial attacks developed: the Fast Gradient Sign Method [9]. This method calculates the adversarial perturbations by taking the gradient of the network's loss with respect to the target image and mapping them to the ℓ_∞ ball.

Given a neural network ($\mathcal{F}$), target sample ($\mathbf{x}$), and label

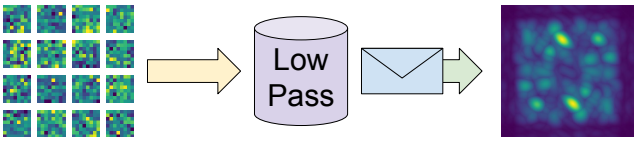


Figure 13. By filtering out the high-frequency components of our filter bank, we also eliminate the high-frequency parts of its spectral envelope, leading to a more adversarially robust model.

(y), the adversarial example $(\tilde{\mathbf{x}})$ is calculated as

$$\tilde{\mathbf{x}} := + \epsilon \cdot \text{sign}(\frac{\partial \mathcal{F}(\mathbf{x}, y)}{\partial \mathbf{x}}) \tag{5}$$

where ϵ is the radius of the ℓ_∞ ball.

This is a simple attack that can be defended against by a variety of methods [3, 4, 9, 18, 20], but it is still a useful tool for analyzing the natural adversarial robustness of an architecture without taking any preventative measures.

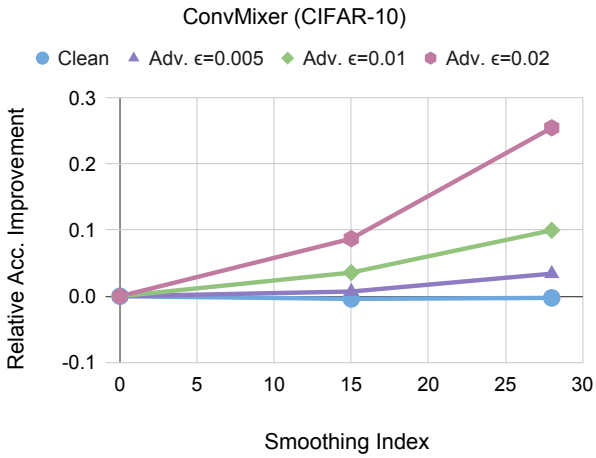


Figure 14. Smoothing the random filters causes our trained models become more robust to adversarial perturbations. We speculate that this is due to removing the high-frequency components of the spectral envelope (Figure 13).

4.3.2 Adversarial Robustness in Separable ResNets

As we see in Figure 10, ResNets with separable convolutions that only learn channel mixing (Sep-Res) are significantly more robust to the adversarial perturbations than the fully-learned separable ResNets. Additionally, it seems as if the static separable networks with lower depth multipliers are even more robust *per parameter*. The deepest Sep-Chans (3) network is nearly just as robust as the deepest Sep-Chans (9) network with only a third of the trainable parameters.

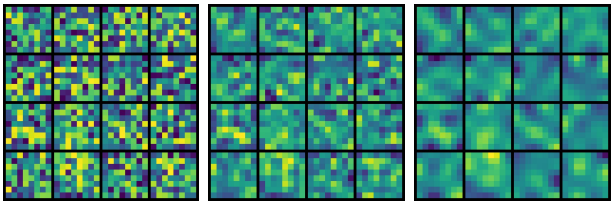


Figure 15. Using smoothed random filters leads to better adversarial accuracy than even fully-leaned networks because they are not susceptible to the high-frequency adversarial noise.

4.3.3 Induced Robustness in ConvMixers

Looking at the learned filters (Figure 12) and their spectrums (Figure 11), one could mistakenly link the *smoothness* of learned filters (relative to that of random filters) to their poor adversarial robustness (Figure 10). Our next experiment shows this to be false.

We can artificially increase the smoothness of our *random* filters by applying a low-pass smoothing operation directly to the filters themselves, as seen in Figure 13. After doing so, the high-frequency components of our random filters have been removed, and the filters now focus more on the mid to low frequencies, just as the learned filters do.

In Figure 14, we see that instead of hurting performance, smoothing the random filters actually significantly *increases* their robustness to adversarial attacks while not hurting performance on clean data (blue line) at all. In fact, we see that smoothing results in a relative performance increase of over 25% for the highest magnitude of attack. We then conclude that there must be a different reason (statistical shortcuts, etc.) for the learned filters' poor robustness.

5. Conclusion

While previous works have shown the merits of fully frozen networks as feature extractors [13, 21], none (to the best of our knowledge) have explored the question of which parameters are the most important or beneficial to learn.

In this work, we have done just that; we motivate theoretically and show empirically that networks that only learn channel mixing can reach nearly the same performance as fully-learned networks without any further alterations. We also show that networks with random spatial-mixing weights are naturally more robust to adversarial attacks, and we also offer a method for *further* increasing the adversarial robustness of such networks.

We hope our work will be of use to the vision community, spurring further questions as to the inner-workings of neural networks and leading to more computationally efficient and adversarially robust models.

6. Acknowledgements

This material is based upon work supported by the National Science Foundation Graduate Research Fellowship under Grant No. DGE1745016.

References

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org. 2

[2] Anonymous. Patches are all you need? In *Submitted to The Tenth International Conference on Learning Representations*, 2022. under review. 1, 2, 3, 4, 7

[3] George Cazenavette, Calvin Murdock, and Simon Lucey. Architectural adversarial robustness: The case for deep pursuit. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7150–7158, 2021. 8

[4] Anirban Chakraborty, Manaar Alam, Vishal Dey, Anupam Chattopadhyay, and Debdeep Mukhopadhyay. Adversarial attacks and defences: A survey, 2018. 8

[5] Mark Chen, Alec Radford, Rewon Child, Jeffrey Wu, Heewoo Jun, David Luan, and Ilya Sutskever. Generative pretraining from pixels. In *International Conference on Machine Learning*, pages 1691–1703. PMLR, 2020. 1, 2, 3

[6] Francois Chollet. Xception: Deep learning with depthwise separable convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017. 2, 3

[7] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009. 5

[8] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*, 2020. 1, 2, 3

[9] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014. 7, 8

[10] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015. 2, 3, 4, 5

[11] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *European conference on computer vision*, pages 630–645. Springer, 2016. 5

[12] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications, 2017. 2, 3

[13] Guang-Bin Huang, Qin-Yu Zhu, and Chee-Kheong Siew. Extreme learning machine: theory and applications. *Neurocomputing*, 70(1-3):489–501, 2006. 1, 2, 8

[14] P. D. Kovesi. MATLAB and Octave functions for computer vision and image processing. Available from: <https://www.peterkovesi.com/matlabfns/>. 11

[15] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009. 5

[16] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25:1097–1105, 2012. 1, 2

[17] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. 1, 2

[18] Yuancheng Li and Yimeng Wang. Defense against adversarial attacks in deep learning. *Applied Sciences*, 9(1), 2019. 8

[19] Stéphane Mallat. *A wavelet tour of signal processing*. Elsevier, 1999. 11

[20] Yaniv Romano, Aviad Aberdam, Jeremias Sulam, and Michael Elad. Adversarial noise attacks of deep learning architectures: Stability analysis via sparse-modeled signals. *Journal of Mathematical Imaging and Vision*, 62(3):313–327, 2020. 8

[21] Andrew M Saxe, Pang Wei Koh, Zhenghao Chen, Maneesh Bhand, Bipin Suresh, and Andrew Y Ng. On random weights and unsupervised feature learning. In *Icml*, 2011. 1, 2, 8

[22] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014. 1, 2

[23] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015. 1, 2

[24] Ilya O. Tolstikhin, Neil Houlsby, Alexander Kolesnikov, Lucas Beyer, Xiaohua Zhai, Thomas Unterthiner, Jessica Yung, Andreas Steiner, Daniel Keysers, Jakob Uszkoreit, Mario Lucic, and Alexey Dosovitskiy. Mlp-mixer: An all-mlp architecture for vision. *CoRR*, abs/2105.01601, 2021. 1, 2, 3

[25] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017. 2

[26] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018. 2, 3

A. Appendix

A.1. Uniqueness of Filters

Ideally, when choosing filters, we want them to obey

$$\min_{i \neq j} \mathcal{E}(h_i, h_j) \quad \text{for} \quad i, j = 1 : k \tag{6}$$

where k is the number of filters and

$$\mathcal{E}(h, g) = \int_x \int_\tau \|h(x) \cdot g(x + \tau)\|_2^2 \cdot dx \cdot d\tau \tag{7}$$

In general, the integral in Equation 7 is hard to evaluate in closed form for arbitrary g. However, if we assume that $g(x) = \sin(\omega \cdot x)$ where ω is an arbitrary frequency, then

$$\begin{aligned} \mathcal{E}_\omega(h) &= \int_x \int_\tau \|h(x) \cdot \sin(\omega \cdot x + \tau)\|_2^2 \cdot dx \cdot d\tau \\ &= \int_x \|h(x) \cdot \sin(\omega \cdot x)\| \cdot dx \\ &\quad + \int_x \|h(x) \cdot \cos(\omega \cdot x)\| \cdot dx \end{aligned} \tag{8}$$

using the property that

$$\sin(\omega \cdot x + \tau) = \cos(\tau) \cdot \sin(\omega \cdot \tau) + \sin(\tau) \cdot \cos(\omega \cdot x) \tag{9}$$

Note that the shift τ is completely integrated out of the expression. Thus, two filters that vary by only a shift will have the same similarity score.

Since this similarity metric is heavily simplified for sinusoidal functions, we can use the *spectrum* as a proxy for it by analyzing the Fourier transforms (decompositions into sine and cosine functions) of our filters.

A.2. 1D Spectral Envelope

The spectral envelope is possibly easier to first understand for 1-dimensional filters. As with the 2D filters in the main text, we obtain the Fourier spectrums by taking the FFT of an 8-long 1D filter padded to be 512 long with the zero frequency shifted to the center of the plot. These spectrums can be viewed on the left side of Figure 16 with the frequencies on the x-axis and magnitudes on the y-axis. Each curve is the spectrum of a different random filter. From top to bottom, we see the spectrums of 1, 4, 16, and 64 filters plotted together.

On the right side of Figure 16, we see the 1D *spectral envelopes* of those same sets of filters. One way to conceptualize the *coverage* of the spectral envelope is the area under the curve. We can see that the spectral coverage of the envelope increases as we add more random filters to our bank.

More information on the spectrums of filters and their envelopes can be found here: [14, 19].

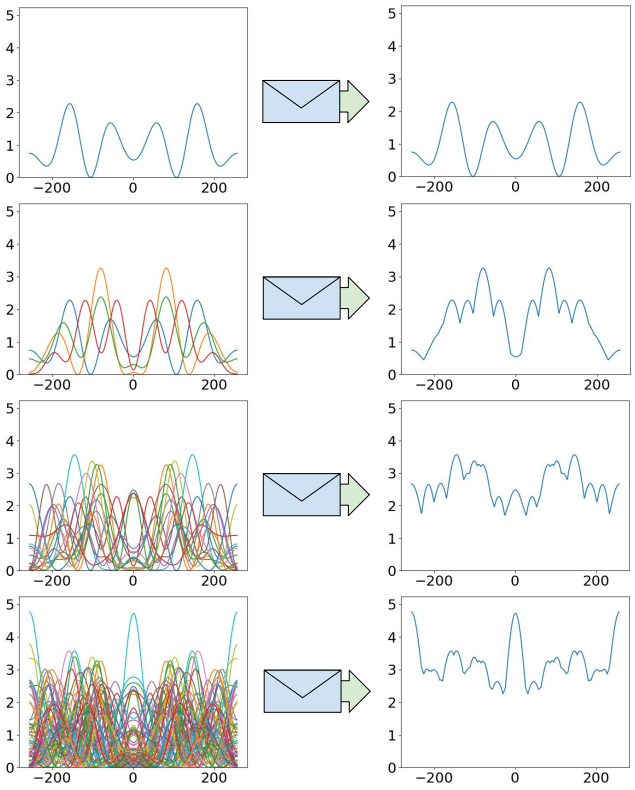


Figure 16. **Left:** Overlays of the individual spectrums of 1, 4, 16, and 64 random 1D filters. **Right:** Spectral envelopes of the same sets of 1, 4, 16, and 64 random filters.

A.3. Filter Smoothing Method

When smoothing large images, a Gaussian blur is often sufficient. However, since our filters are much smaller than images, the boundary effects caused by adding the padding for a blurring convolution would be much more impactful by comparison.

Instead, we choose a more direct method of smoothing by projecting our filters into their discrete cosine transform (DCT) bases. After this projection, we can induce a smoothing effect by directly zeroing our the higher-order components. The "smoothing index" referenced in the main text is the number of higher-order components removed. Since we are performing a 2D DCT on an 8×8 filter, keeping just the first 7×7 components is equivalent to removing the last 15. Likewise, keeping only the first 6×6 is equivalent to removing the last 28.