A Learnable ScatterNet: Locally Invariant Convolutional Layers

1 Abstract

In this section we explore tying together the ideas from Scattering Transforms and Convolutional Neural Networks (CNN) for Image Analysis by proposing a learnable ScatterNet. Previous attempts at tying them together in hybrid networks have tended to keep the two parts separate, with the ScatterNet forming a fixed front end and a CNN forming a learned backend. We instead look at adding learning between scattering orders, as well as adding learned layers before the ScatterNet. We do this by breaking down the scattering orders into single convolutional-like layers we call 'locally invariant' layers, and adding a learned mixing term to this layer. Our experiments show that these locally invariant layers can improve accuracy when added to either a CNN or a ScatterNet. We also discover some surprising results in that the ScatterNet may be best positioned after one or more layers of learning rather than at the front of a neural network.

2 Introduction

In image understanding tasks such as classification, recognition and segmentation, Convolutional Neural Networks (CNNs) have now become the *de facto* and the state of the art model. Since they proved their worth in 2012 by winning the ImageNet Large Scale Visual Recognition Competition (ILSVRC) [19] with the AlexNetwork [11], they have been fine tuned and developed into very powerful and flexible models. Most of this development has been in changing the architecture, such as the Residual Network[7], the Inception Network [25] and the DenseNet [9]. However one of the key building blocks of CNNs, the convolutional filter bank, has seen less development and in today's models they are not too dissimilar to what they were in 2012. We believe there is still much to explore in the way convolutional filters are built and learned.

Current layers are randomly initialized and learned through backpropagation by minimizing a custom loss function. The properties and purpose of learned filters past the first layer of a CNN are not well understood, a deeply unsatisfying situation, particularly when we start to see problems in modern solutions such as significant redundancy [6] and weakness to adversarial attacks [2].

The Scattering Transform by Mallat et. al. [14, 1] attempts to address the problems of poorly understood filtering layers by using predefined wavelet bases whose properties are well known. Using this knowledge, Mallat derives bounds on the effect of noise and deformations to the input. This work inspires us, but the fixed nature of ScatterNets has proved a limiting factor for them so far.

To combat this, [16, 15, 21] use ScatterNets as a front end for deep learning tasks, calling them Hybrid ScatterNets. These have had some success, but in [5] we visualized what features a ScatterNet extracted and showed that they were ripple-like and very dissimilar from what a CNN would extract, suggesting that some learning should be done between the ScatterNet orders.

To do this, we take inspiration from works like [17, 10, 26, 4], which decompose convolutional filters as a learned mixing of fixed harmonic functions. However we now propose to build a convolutional-like layer which is a learned mixing of the locally invariant scattering terms from the original ScatterNet [1].

In section 3 we briefly review convolutional layers and scattering layers before introducing our learnable scattering layers in section 4. In section 5 we describe how we implement our proposed layer, and present some experiments we have run in section 6 and then draw conclusions about how these new ideas might improve neural networks in the future.

3 Related Work

3.1 Convolutional Layers

Let the output of a CNN at layer l be:

$$x^{(l)}(c,\mathbf{u}), \quad c\in\{0,\ldots C_l-1\}, \mathbf{u}\in\mathbb{R}^2$$

where c indexes the channel dimension , and $\mathbf{u}$ is a vector of coordinates for the spatial position. Of course, $\mathbf{u}$ is typically sampled on a grid, but following the style of [17], we keep it continuous to more easily differentiate between the spatial and channel dimensions. A typical convolutional layer in a standard CNN (ignoring the bias term) is:

$$y^{(l+1)}(f,\mathbf{u}) = \sum_{c=0}^{C_l-1} x^{(l)}(c,\mathbf{u}) * h_f^{(l)}(c,\mathbf{u}) \tag{3.1}$$

$$x^{(l+1)}(f,\mathbf{u}) = \sigma\left(y^{(l+1)}(f,\mathbf{u})\right) \tag{3.2}$$

where $h_f^{(l)}(c,\mathbf{u})$ is the fth filter of the lth layer (i.e. $f\in\{0,\ldots,C_{l+1}-1\}$) with C_l different point spread functions. σ is a non-linearity possibly combined with scaling such as batch normalization. The convolution is done independently for each c in the C_l channels and the resulting outputs are summed together to give one activation map. This is repeated C_{l+1} times to give $\left\{x^{(l+1)}(f,\mathbf{u})\right\}_{f\in\{0,\ldots,C_{l+1}-1\},\mathbf{u}\in\mathbb{R}^2}$

3.2 Wavelets and Scattering Transforms

The 2-D wavelet transform is done by convolving the input with a mother wavelet dilated by 2^j and rotated by θ:

$$\psi_{j,\theta}(\mathbf{u}) = 2^{-j}\psi\left(2^{-j}R_{-\theta}\mathbf{u}\right) \tag{3.3}$$

where R is the rotation matrix, $1\leq j\leq J$ indexes the scale, and $1\leq k\leq K$ indexes θ to give K angles between 0 and π. We copy notation from [1] and define $\lambda=(j,k)$ and the set of all possible λs is Λ whose size is $|\Lambda|=JK$. The wavelet transform, including lowpass, is then:

$$Wx(c,\mathbf{u}) = \{x(c,\mathbf{u})*\phi_J(\mathbf{u}), x(c,\mathbf{u})*\psi_\lambda(\mathbf{u})\}_{\lambda\in\Lambda} \tag{3.4}$$

Taking the modulus of the wavelet coefficients removes the high frequency oscillations of the output signal while preserving the energy of the coefficients over the frequency band covered by ψ_λ. This is crucial to ensure that the scattering energy is concentrated towards zero-frequency as the scattering order increases, allowing sub-sampling. We define the wavelet modulus propagator to be:

$$\tilde{W}x(c,\mathbf{u}) = \{x(c,\mathbf{u})*\phi_J(\mathbf{u}), |x(c,\mathbf{u})*\psi_\lambda(\mathbf{u})|\}_{\lambda\in\Lambda} \tag{3.5}$$

Let us call these modulus terms $U[\lambda]x = |x * \psi_\lambda|$ and define a path as a sequence of λs given by $p = (\lambda_1, \lambda_2, \dots \lambda_m)$. Further, define the modulus propagator acting on a path p by:

$$\begin{aligned} U[p]x &= U[\lambda_m] \cdots U[\lambda_2] U[\lambda_1] x & (3.6) \\ &= || \cdots |x * \psi_{\lambda_1}| * \psi_{\lambda_2}| \cdots * \psi_{\lambda_m}| & (3.7) \end{aligned}$$

These descriptors are then averaged over the window 2^J by a scaled lowpass filter $\phi_J = 2^{-J}\phi(2^{-J}\mathbf{u})$ giving the 'invariant' scattering coefficient

$$S[p]x(\mathbf{u}) = U[p]x * \phi_J(\mathbf{u}) \tag{3.8}$$

If we define $p + \lambda = (\lambda_1, \dots, \lambda_m, \lambda)$ then we can combine Equation 3.5 and Equation 3.6 to give:

$$\tilde{W}U[p]x = \{S[p]x, U[p+\lambda]x\}_\lambda \tag{3.9}$$

Hence we iteratively apply $\tilde{W}$ to all of the propagated U terms of the previous layer to get the next order of scattering coefficients and the new U terms.

The resulting scattering coefficients have many nice properties, one of which is stability to diffeomorphisms (such as shifts and warping). From [14], if $\mathcal{L}_\tau x = x(\mathbf{u} - \tau(\mathbf{u}))$ is a diffeomorphism which is bounded with $\|\nabla \tau\|_\infty \leq 1/2$, then there exists a $K_L > 0$ such that:

$$\|S\mathcal{L}_\tau x - Sx\| \leq K_L P H(\tau) \|x\| \tag{3.10}$$

where $P = \text{length}(p)$ is the scattering order, and $H(\tau)$ is a function of the size of the displacement, derivative and Hessian of τ.

4 Locally Invariant Layer

The U terms in subsection 3.2 are often called 'covariant' terms but in this paper we will call them locally invariant, as they tend to be invariant up to a scale 2^j. We propose to mix the locally invariant terms U and the lowpass terms S with learned weights $a_{f,\lambda}$ and b_f. For example, consider a first order ScatterNet, and let the input to it be $x^{(l)}$. Our proposed output $y^{(l+1)}$ is then:

$$\begin{aligned} y^{(l+1)}(f, \mathbf{u}) &= \sum_{\lambda \in \Lambda} \sum_{c=0}^{C-1} |x^{(l)}(c, \mathbf{u}) * \psi_\lambda(\mathbf{u})| \; a_{f,\lambda}(c) \\ &+ \left(\sum_{c=0}^{C-1} x^{(l)}(c, \mathbf{u}) * \phi_J(\mathbf{u}) \right) b_f(c) & (4.1) \end{aligned}$$

Returning to Equation 3.5, we define a new index variable γ such that $\tilde{W}[\gamma]x = x * \phi_J$ for $\gamma = 1$ and $\tilde{W}[\gamma]x = |x * \psi_\lambda|$ for $2 \leq \gamma \leq JK + 1$. We do the same for the weights a, b by defining $\tilde{a}_f = \left\{b_f, a_{f,\lambda}\right\}_\lambda$ and $\tilde{a}_f[\gamma] = b_f$ if $\gamma = 1$ and $\tilde{a}_f[\gamma] = a_{f,\lambda}$ if $2 \leq \gamma \leq JK + 1$. We further define $q = (c, \gamma) \in Q$ to combine the channel and index terms. This simplifies Equation 4.1 to be:

$$\begin{aligned} z^{(l+1)}(q, \mathbf{u}) &= \tilde{W}x^{(l)}[q] = \tilde{W}x^{(l)}[\tilde{\lambda}](c, \mathbf{u}) & (4.2) \\ y^{(l+1)}(f, \mathbf{u}) &= \sum_{q \in Q} z^{(l+1)}(q, \mathbf{u}) \tilde{a}_f(q) & (4.3) \end{aligned}$$

or in matrix form with $A_{f,q} = \tilde{a}_f(q)$

$$Y^{(l+1)}(\mathbf{u}) = AZ^{(l+1)}(\mathbf{u}) \tag{4.4}$$

This is very similar to the standard convolutional layer from Equation 3.1, except we have replaced the previous layer's x with intermediate coefficients z (with $|Q| = (JK+1)C$ channels), and the convolutions of Equation 3.1 have been replaced by a matrix multiply (which can also be seen as a 1×1 convolutional layer). We can then apply Equation 3.2 to Equation 4.3 to get the next layer's output (or equivalently, the next order scattering coefficients).

4.1 Properties

The first thing to note is that with careful choice of A and σ, we can recover the original translation invariant ScatterNet [1, 16]. If $C_{l+1} = (JK+1)C_l$ and A is the identity matrix $I_{C_{l+1}}$, we remove the mixing and then $y^{(l+1)} = \tilde{W}x$.

Further, if $\sigma = \text{ReLU}$ as is commonly the case in training CNNs, it has no effect on the positive locally invariant terms U. It will affect the averaging terms if the signal is not positive, but this can be dealt with by adding a channel dependent bias term α_c to $x^{(l)}$ to ensure it is positive. This bias term will not affect the propagated signals as $\int \alpha_c \psi_\lambda(\mathbf{u}) d\mathbf{u} = 0$. The bias can then be corrected by subtracting $\alpha_c \|\phi_J\|_2$ from the averaging terms after taking the ReLU, then $x^{(l+1)} = \tilde{W}x$.

This makes one layer of our system equivalent to a first order scattering transform, giving S_0 and U_1 (invariant to input shifts of 2^1). Repeating the same process for the next layer again works, as we saw in Equation 3.6, giving S_1 and U_2 (invariant to shifts of 2^2). If we want to build higher invariance, we can continue or simply average these outputs with an average pooling layer.

Let us define the action of our layer on the scattering coefficients to be Vx. We would like to find a bound on $\|V\mathcal{L}_\tau x - Vx\|$. To do this, we note that the mixing is a linear operator and hence is Lipschitz continuous. The authors in [17] find constraints on the mixing weights to make them non-expansive (i.e. Lipschitz constant 1). Further, the ReLU is non-expansive meaning the combination of the two is also non-expansive, so $\|V\mathcal{L}_\tau x - Vx\| \leq \|S\mathcal{L}_\tau x - Sx\|$, and Equation 3.10 holds.

5 Implementation

Like [23, 22] we use the DT$\mathbb{C}$WT [20] for our wavelet filters $\psi_{j,\theta}$ due to their fast implementation with separable convolutions which we will discuss more in subsection 5.2. A side effect of this choice is that the number of orientations of wavelets is restricted to $K = 6$.

The output of the DT$\mathbb{C}$WT is decimated by a factor of 2^j in each direction for each scale j. In all our experiments we set $J = 1$ for each invariant layer, meaning we can mix the lowpass and bandpass coefficients at the same resolution. Figure 1 shows how this is done. Note that setting $J = 1$ for a single layer does not restrict us from having $J > 1$ for the entire system, as if we have a second layer with $J = 1$ after the first, including downsampling ($\downarrow$), we would have:

$$\left(((x * \phi_1) \downarrow 2) * \psi_{1,\theta}\right) \downarrow 2 = \left(x * \psi_{2,\theta}\right) \downarrow 4 \tag{5.1}$$

5.1 Memory Cost

A standard convolutional layer with C_l input channels, C_{l+1} output channels and kernel size L has $L^2 C_l C_{l+1}$ parameters.

The number of learnable parameters in each of our proposed invariant layers with $J = 1$ and $K = 6$ orientations is:

$$\#\text{params} = (JK+1)C_l C_{l+1} = 7C_l C_{l+1} \tag{5.2}$$

The spatial support of the wavelet filters is typically 5×5 pixels or more, and we have reduced #params to less than 3×3 per filter, while producing filters that are significantly larger than this.

5.2 Computational Cost

A standard convolutional layer with kernel size L needs $L^2 C_{l+1}$ multiplies per input pixel (of which there are $C_l \times H \times W$).

As mentioned in subsection 5.1, we use the DT$\mathbb{C}$WT for our complex, shift invariant wavelet decomposition. We use the open source Pytorch implementation of the DT$\mathbb{C}$WT [3] as it can run on GPUs and has support for backpropagating gradients.

There is an overhead in doing the wavelet decomposition for each input channel. A regular discrete wavelet transform (DWT) with filters of length L will have $2L\left(1-2^{-2J}\right)$ multiplies for a J scale decomposition. A DT$\mathbb{C}$WT has 4 DWTs for a 2-D input, so its cost is $8L\left(1-2^{-2J}\right)$, with $L = 6$ a common size for the filters. It is important to note that unlike the filtering operation, this does not scale with C_{l+1}, the end result being that as C_{l+1} grows, the cost of C_l forward transforms is outweighed by that of the mixing process.

Because we are using a decimated wavelet decomposition, the sample rate decreases after each wavelet layer. The benefit of this is that the mixing process then only works on one quarter the spatial size after one first scale and one sixteenth the spatial after the second scale. Restricting ourselves to $J = 1$ as we mentioned in section 5, the computational cost is then:

$$\underbrace{\frac{7}{4}C_{l+1}}_{\text{mixing}} + \underbrace{36}_{\text{DT}\mathbb{C}\text{WT}} \quad \text{multiplies per input pixel} \tag{5.3}$$

In most CNNs, C_{l+1} is several dozen if not several hundred, which makes Equation 5.3 significantly smaller than $L^2 C_{l+1} = 9C_{l+1}$ multiplies for 3×3 convolutions.

6 Experiments

In this section we examine the effectiveness of our invariant layer by testing its performance on the well known datasets CIFAR-10 (10 classes, 5000 images per class at 32×32 pixels per image), CIFAR-100 (100 classes, 500 images per class at 32×32 pixels per image) and Tiny ImageNet[12] (a dataset like ImageNet with 200 classes and 500 training images per class, each image at 64×64 pixels). Our experiment code is available at https://github.com/fbcotter/invariant_convolution.

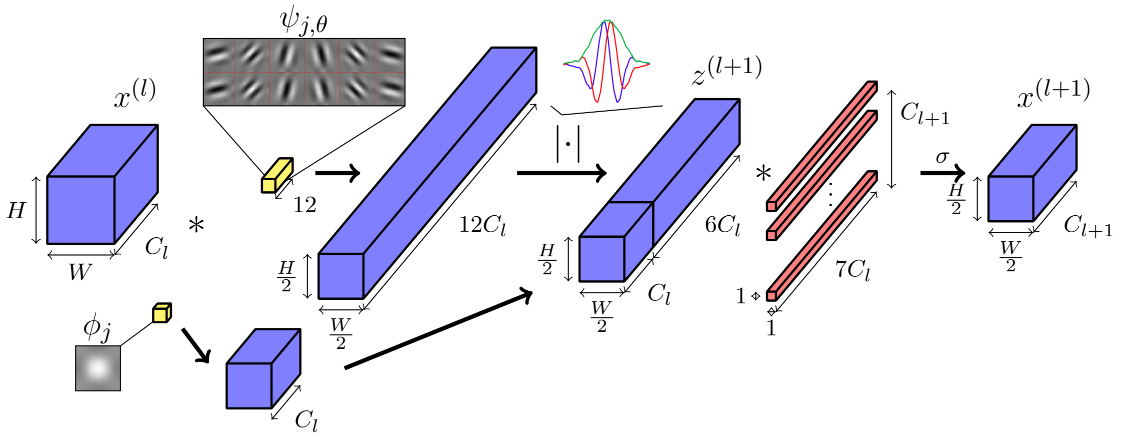

Figure 1: Block Diagram of Proposed Invariant Layer for $J = 1$. Activations are shaded blue, fixed parameters yellow and learned parameters red. Input $x^{(l)} \in \mathbb{R}^{C_l \times H \times W}$ is filtered by real and imaginary oriented wavelets and a lowpass filter and is downsampled. The channel dimension increases from C_l to $(2K+1)C_l$, where the number of orientations is $K = 6$. The real and imaginary parts are combined by taking their magnitude (an example of what this looks like in 1D is shown above the magnitude operator) - the components oscillating in quadrature are combined to give $z^{(l+1)}$. The resulting activations are concatenated with the lowpass filtered activations, mixed across the channel dimension, and then passed through a nonlinearity σ to give $x^{(l+1)}$. If the desired output spatial size is $H \times W$, $x^{(l+1)}$ can be bilinearly upsampled paying only a few multiplies per pixel.

6.1 Layer Level Comparison

To compare our proposed locally invariant layer (inv) to a regular convolutional layer (conv), we use a simple yet powerful VGG-like architecture with 6 convolutional layers for CIFAR and 8 layers for Tiny ImageNet as shown in Table 1. The initial number of channels C we use is 64. Despite this simple design, this reference architecture achieves competitive performance for the three datasets.

This network is optimized with stochastic gradient descent with momentum. The initial learning rate is 0.5, momentum is 0.85, batch size $N = 128$ and weight decay is 10^{-4}. For CIFAR-10/CIFAR-100 we scale the learning rate by a factor of 0.2 after 60, 80 and 100 epochs, training for 120 epochs in total. For Tiny ImageNet, the rate change is at 18, 30 and 40 epochs (training for 45 in total).

We perform an ablation study where we progressively swap out convolutional layers for invariant layers keeping the input and output activation sizes the same. As there are 6 layers (or 8 for Tiny ImageNet), there are too many permutations to list the results for swapping out all layers for our locally invariant layer, so we restrict our results to swapping 1 or 2 layers. Table 2 reports the top-1 classification accuracies for CIFAR-10, CIFAR-100 and Tiny ImageNet. In addition to testing on the full datasets we report results for a reduced training set size. In the table, ‘invX’ means that the ‘convX’ layer from Table 1 is replaced with an invariant layer.

Interestingly, we see improvements when one or two invariant layers are used near the start of a system, but not for the first layer.

Table 1: Base Architecture used for Convolution Layer comparison tests. †indicates only used in Tiny ImageNet experiments.

Name	**Output Size**
convA	$C \times H \times W$
convB	$C \times H \times W$
convC	$2C \times H/2 \times W/2$
convD	$2C \times H/2 \times W/2$
convE	$4C \times H/4 \times W/4$
convF	$4C \times H/4 \times W/4$
convG†	$8C \times H/8 \times W/8$
convH†	$8C \times H/8 \times W/8$
fc	num classes

6.2 Network Comparison

In the previous section, we examined how the locally invariant layer performs when directly swapped in for a convolutional layer in an architecture designed for convolutional layers. In this section, we look at how it performs in a Hybrid ScatterNet-like [15, 16], network.

To build the second order ScatterNet, we stack two invariant layers on top of each other. For 3 input channels, the output of these layers has $3(1+6+6+36) = 147$ channels at $1/16$ the spatial input size. We then use 4 convolutional layers, similar to convC to convF in Table 1 with $C = 96$. In addition, we use dropout after these later convolutional layers with drop probability $p = 0.3$.

We compare a ScatterNet with no learning in between scattering orders (ScatNet A) to one with our proposal for a learned mixing matrix A (ScatNet B). Finally, we also test the hypothesis seen from subsection 6.1 about putting conv layers before an inv layer, and test a version with a small convolutional layer before ScatNets A and B, taking the input from 3 to 16 channels, and call these ScatNet architectures ScatNet C and D respectively.

See Table 3 for results from these experiments. It is clear from the improvements that the mixing layer helps the Scattering front end. Interestingly, ScatNet C and D further improve on the A and B versions (albeit with a larger parameter and multiply cost than the mixing operation). This reaffirms that there may be benefit to add learning before as well as inside the ScatterNet.

For comparison, we have also listed the performance of other architectures as reported by their authors in order of increasing complexity. Our proposed ScatNet D achieves comparable performance with the the All Conv, VGG16 and FitNet architectures. The Deep[8] and Wide[27] ResNets perform best, but with very many more multiplies, parameters and layers.

ScatNets A to D with 6 layers like convC to convG from Table 1 after the scattering, achieve $58.1, 59.6, 60.8$ and 62.1% top-1 accuracy on Tiny ImageNet. As these have more parameters and multiplies from the extra layers we exclude them from Table 3.

7 Conclusion

In this work we have proposed a new learnable scattering layer, dubbed the locally invariant convolutional layer, tying together ScatterNets and CNNs. We do this by adding a mixing between the layers of ScatterNet allowing the learning of more complex shapes than the ripples seen in [5].

Table 2: Results for testing VGG like architecture with convolutional and invariant layers on several datasets. An architecture with 'invX' means the equivalent convolutional layer 'convX' from Table 1 was swapped for our proposed layer.

	CIFAR-10		CIFAR-100		Tiny ImgNet	
Trainset Size	10k	50k	10k	50k	20k	100k
ref	84.4	91.9	53.2	70.3	37.4	59.1
invA	82.8	91.3	48.4	69.5	33.2	57.7
invB	84.8	91.8	54.8	70.7	36.9	59.5
invC	85.3	92.3	54.2	71.2	37.1	59.8
invD	83.8	91.2	51.2	70.1	37.6	59.3
invE	83.3	91.6	50.3	70.0	37.8	59.4
invF	82.1	90.5	47.6	68.9	34.0	57.8
invA, invB	83.1	90.5	49.8	68.4	35.0	57.9
invB, invC	83.8	91.2	50.6	69.1	34.6	57.5
invC, invD	85.1	92.1	54.3	70.1	**37.9**	59.0
invD, invE	80.2	89.1	49.0	67.3	33.9	57.5
invA, invC	82.8	90.7	49.5	69.6	34.0	56.9
invB, invD	**85.4**	**92.7**	**54.6**	**71.3**	**37.9**	59.8
invC, invE	84.8	91.8	53.5	70.9	37.6	**60.2**

Table 3: Hybrid ScatterNet top-1 classification accuracies on CIFAR-10 and CIFAR-100. N_l is the number of learned convolutional layers, #param is the number of parameters, and #mults is the number of multiplies per $32 \times 32 \times 3$ image. An asterisk indicates that the value was estimated from the architecture description.

	N_l	#param	#mults	CIFAR-10	CIFAR-100
ScatNet A	4	2.6M	165M	89.4	67.0
ScatNet B	6	2.7M	167M	91.1	70.7
ScatNet C	5	3.7M	251M	91.6	70.8
ScatNet D	7	4.3M	294M	93.0	73.5
All Conv[24]	8	1.4M	281M*	92.8	66.3
VGG16[13]	16	138M*	313M*	91.6	-
FitNet[18]	19	2.5M	382M	91.6	65.0
ResNet-1001[8]	1000	10.2M	4453M*	95.1	77.3
WRN-28-10[27]	28	36.5M	5900M*	96.1	81.2

This invariant layer can easily be shaped to allow it to drop in the place of a convolutional layer, theoretically saving on parameters and computation. However, care must be taken when doing this, as our ablation study showed that the layer only improves upon regular convolution at certain depths. Typically, it seems wise to interleave convolutional layers and invariant layers.

We have developed a system that allows us to pass gradients through the Scattering Transform, something that previous work has not yet researched. Because of this, we were able to train end-to-end a system that has a ScatterNet surrounded by convolutional layers and with our proposed mixing. We were surprised to see that even a small convolutional layer before Scattering helps the network, and a very shallow and simple Hybrid-like ScatterNet was able to achieve good performance on CIFAR-10 and CIFAR-100.

There is still much research to do - why does the proposed layer work best near, but not at the beginning of deeper networks? Why is it beneficial to precede an invariant layer with a convolutional layer? Can we combine invariant layers in Residual style architectures? The work presented here is still nascent but we hope that it will stimulate further interest and research into both ScatterNets and the design of convolutional layers in CNNs.

8 More to Come

Need to look at what the first layer of the ScatNet D system looks like.

Need to do some analysis on the gradient of a magnitude operation (not previously done).

Can look at the structure of the A matrix.

References

[1] J. Bruna and S. Mallat. "Invariant Scattering Convolution Networks". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 35.8 (Aug. 2013), pp. 1872–1886.

[2] N. Carlini and D. Wagner. "Towards Evaluating the Robustness of Neural Networks". In: *2017 IEEE Symposium on Security and Privacy (SP)*. May 2017, pp. 39–57.

[3] Fergal Cotter. *Pytorch Wavelets*. GitHub fbcotter/pytorch_wavelets, 2018.

[4] Fergal Cotter and Nick Kingsbury. "Deep Learning in the Wavelet Domain". In: *arXiv:1811.06115 [cs]* (Nov. 2018). arXiv: 1811.06115 [cs].

[5] Fergal Cotter and Nick Kingsbury. "Visualizing and Improving Scattering Networks". In: *2017 IEEE 27th International Workshop on Machine Learning for Signal Processing (MLSP)*. Sept. 2017, pp. 1–6. arXiv: 1709.01355.

[6] Emily Denton et al. "Exploiting Linear Structure Within Convolutional Networks for Efficient Evaluation". In: *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 1*. NIPS'14. Cambridge, MA, USA: MIT Press, 2014, pp. 1269–1277.

[7] Kaiming He et al. "Deep Residual Learning for Image Recognition". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2016, pp. 770–778. arXiv: 1512.03385.

[8] Kaiming He et al. "Identity Mappings in Deep Residual Networks". In: *arXiv:1603.05027 [cs]* (Mar. 2016). arXiv: 1603.05027 [cs].

[9] Gao Huang et al. “Densely Connected Convolutional Networks”. In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. July 2017, pp. 2261–2269. arXiv: 1608.06993.

[10] Jörn-Henrik Jacobsen, Bert de Brabandere, and Arnold W. M. Smeulders. “Dynamic Steerable Blocks in Deep Residual Networks”. In: *arXiv:1706.00598 [cs, stat]* (June 2017). arXiv: 1706.00598 [cs, stat].

[11] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. “ImageNet Classification with Deep Convolutional Neural Networks”. In: *NIPS*. Curran Associates, Inc., 2012, pp. 1097–1105.

[12] Fei-Fei Li. “Tiny ImageNet Visual Recognition Challenge”. In: Stanford cs231n: https://tiny-imagenet.herokuapp.com/.

[13] S. Liu and W. Deng. “Very Deep Convolutional Neural Network Based Image Classification Using Small Training Sample Size”. In: *2015 3rd IAPR Asian Conference on Pattern Recognition (ACPR)*. Nov. 2015, pp. 730–734.

[14] Stéphane Mallat. “Group Invariant Scattering”. en. In: *Communications on Pure and Applied Mathematics* 65.10 (Oct. 2012), pp. 1331–1398.

[15] Edouard Oyallon. “A Hybrid Network: Scattering and Convnet”. In: 2017.

[16] Edouard Oyallon, Eugene Belilovsky, and Sergey Zagoruyko. “Scaling the Scattering Transform: Deep Hybrid Networks”. In: *2017 IEEE International Conference on Computer Vision (ICCV)*. Oct. 2017, pp. 5619–5628. arXiv: 1703.08961.

[17] Qiang Qiu et al. “DCFNet: Deep Neural Network with Decomposed Convolutional Filters”. In: *arXiv:1802.04145 [cs, stat]* (Feb. 2018). arXiv: 1802.04145 [cs, stat].

[18] Adriana Romero et al. “FitNets: Hints for Thin Deep Nets”. In: *arXiv:1412.6550 [cs]* (Dec. 2014). arXiv: 1412.6550 [cs].

[19] Olga Russakovsky et al. “ImageNet Large Scale Visual Recognition Challenge”. In: *arXiv:1409.0575 [cs]* (Sept. 2014). arXiv: 1409.0575 [cs].

[20] Ivan W. Selesnick, Richard G. Baraniuk, and Nick G. Kingsbury. “The Dual-Tree Complex Wavelet Transform”. In: *Signal Processing Magazine, IEEE* 22.6 (2005), pp. 123–151.

[21] Amarjot Singh. “ScatterNet Hybrid Frameworks for Deep Learning”. PhD thesis. University of Cambridge, May 2018.

[22] Amarjot Singh and Nicholas Kingsbury. “Multi-Resolution Dual-Tree Wavelet Scattering Network for Signal Classification”. en. In: *11th International Conference on Mathematics in Signal Processing*. Birmingham, UK, Dec. 2016.

[23] Amarjot Singh and Nick Kingsbury. “Dual-Tree Wavelet Scattering Network with Parametric Log Transformation for Object Classification”. In: *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. Mar. 2017, pp. 2622–2626. arXiv: 1702.03267.

[24] Jost Tobias Springenberg et al. “Striving for Simplicity: The All Convolutional Net”. In: *arXiv:1412.6806 [cs]* (Dec. 2014). arXiv: 1412.6806 [cs].

[25] Christian Szegedy et al. “Going Deeper With Convolutions”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2015, pp. 1–9.

[26] Daniel E. Worrall et al. “Harmonic Networks: Deep Translation and Rotation Equivariance”. en. In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Honolulu, HI: IEEE, July 2017, pp. 7168–7177.

[27] Sergey Zagoruyko and Nikos Komodakis. “Wide Residual Networks”. In: *arXiv:1605.07146 [cs]* (May 2016). arXiv: 1605.07146 [cs].