Build the noise, shift, deformation stable filter group based on the reverse engineering with Scattering Transform

Authors

In this paper, we have used the neural network as the tool to implement the 2D Scattering Transform. Previous studies of the Scattering Transform have calculated the convolution using the fast Fourier transform and try to integrate them into some deep learning frameworks. Our works explore tying together the ideas from Scattering Transform and convolutional neural networks to capture the best features. Firstly, we design the best suitable neural network architecture which its output is corresponds to the output of Scattering Transform. In the next step, we propose the process for training the designed model and extract the kernel weights. We name this kernel weights is the noise, shift, deformation stable filter group. Finally, we initialize them into the convolutional neural layers will be equivalent to Scattering Transform. This approach allows us easily achieve the nice properties of Scattering Transform without depending on any deep learning frameworks. It will accelerate Scattering Transform and energy-efficient.

Introduction: The Scattering Transform [1] was developed by Stephane Mallat and Joan Bruna in 2013. It is a widely-used signal presentation and has become a suitable choice as feature extractors in many computer vision applications. It was built using wavelet multiscale decompositions with a deep convolutional architecture. The Scattering Transform design was based on the strong mathematical foundations [2], leading to the loss of information is avoided and energy-preserving.

The Scattering Transform captures the key geometrical properties. It has many nice properties that are stability to additive noise, shifts, small deformation [1]. These variabilities have been vanished that will be efficient in the image classification tasks.

Scattering Transform had proved its significant effect over the past decade and hitherto. It is used in a variety of applications with many data types such as relating to images, audio recordings, and electronic densities. Besides, it's also integrated into many modern learning models, including hybrid CNN architecture, Generative Adversarial Network model, graph model, inverse problems [5, 6, 8].

The first demonstrated in [1] is state-of-the-art classification results for handwritten digits and texture discrimination. In [3], Scattering transform is applied to build representations for quantum chemistry and leads to state-of-the-art performance in the regression of molecular energies. In [4], Scattering Transform is also extended to analyze graph data. Scattering based classification models have been further developed in [5, 6], by integrating them within CNN architectures as preprocessing stages and the learned mixing [6]. The results from [5] demonstrate the effective combination with modern representation learning approaches. The most recent time, the package *Kymatio* [7] and *Pytorch Wavelets* [8] provide a modern implementation of Scattering Transform leveraging efficient GPU-optimized routines.

At the moment, there are two difficult problems when applying Scattering Transform. The complexity in designing the pre-defined filter banks leads to slow execution and the Fourier-based convolution method leads the limits in backend support and flexibility [7]. Our work solves two of these weaknesses. Our method based on reverse engineering with Scattering Transform and the convolutional technique in the neural networks. We have built an invariant stable filter group and initialize them to any convolutional neural networks. This filter group makes optimization for speed, flexibility, and backend support.

Our key task focuses on designing the neural model which is equivalent to the *first-order* and *second-order* Scattering Transform. Besides, we also propose the diagram for learning the feature from Scattering Transform. Next, we train the designed neural model and optimizing the loss function between two outputs of the Scattering Transform and our model. The final step will extract the kernel weights from the learned model and inject them into the convolutional layers.

In the next section, we briefly introduce the Scattering Transform, describe how implements our method, and some results.

Scattering Transform Theory: The backend of the Scattering Transform is FFT-based [7]. It is defined as a complex-valued convolutional network which filters are fixed to be the complex wavelets coupled with modulus non-linearities and low-pass filters. Each layer is a wavelet transform, which separates the scales of the incoming signal. Notating some symbols in convolution, with $\star$ denotes convolution, c indexes the channel dimension, u is a vector of coordinates for the spatial position, $x(c, \mathbf{u})$ is receptive field, $\phi_J = 2^{-J}\phi\left(2^{-J}\mathbf{u}\right)$ is a scaled low-pass filter.

Let us consider set of wavelets $\{\psi_\lambda\}_\lambda$ is done by convolving the input with a mother wavelet dilated by 2^j and rotated by θ:

$$\psi_{j,\theta}(\mathbf{u}) = 2^{-j}\psi\left(2^{-j}R_{-\theta}\mathbf{u}\right) \tag{1}$$

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 θ and π.

The *complex Morlet wavelet* ψ is used in implementing Scattering Transform. Its real and imaginary parts are nearly quadrature phase filters. The complex-value Morlet wavelet is given by

$$\psi(u) = \alpha\left(e^{iu.\xi} - \beta\right)e^{-|u|^2/(2\sigma^2)} \tag{2}$$

The complex modulus non-linear operator $|y|$ over complex signals $y = y_r + iy_i$, is calculated:

$$|y(u)| = \left(|y_r(u)|^2 + |y_i(u)|^2\right)^{1/2} \tag{3}$$

Defining $\lambda = (j, k)$ and the set of all possible λ_s is Λ whose size is $|\Lambda| = JK$. The wavelet transform, including low-pass, is then:

$$Wx(c, \mathbf{u}) = \{x(c, \mathbf{u}) \star \phi_J(\mathbf{u}),\ x(c, \mathbf{u}) \star \psi_\lambda(\mathbf{u})\}_{\lambda \in \Lambda} \tag{4}$$

To remove the high frequency oscillation of the output signal while preserving energy of the coefficients over the frequency bands covered by ψ_λ, we apply a point-wise complex modulus non-linear operator to $x(c, \mathbf{u}) \star \psi_\lambda(\mathbf{u})$, allowing by an appropriate downsampling. It is necessary to a translation invariant representation. We define the wavelet modulus propagator to be:

$$\tilde{W}x(c, \mathbf{u}) = \{x(c, \mathbf{u}) \star \phi_J(\mathbf{u}),\ |x(c, \mathbf{u}) \star \psi_\lambda(\mathbf{u})|\}_{\lambda \in \Lambda} \tag{5}$$

Let us name $U[\lambda]x = |x \star \psi_\lambda|$ is the modulus terms. Any sequence $p = (\lambda_1, \lambda_2, ..., \lambda_m)$ defines a $path$ which is computed an ordered product of convolution and non-linear operators:

$$\begin{aligned} U[p]x &= U\left[\lambda_m\right] \cdots U\left[\lambda_2\right] U\left[\lambda_1\right] x \\ &= \left|\left|\cdots\left|x \star \psi_{\lambda_1}\right| \star \psi_{\lambda_2}\right| \cdots \star \psi_{\lambda_m}\right| \end{aligned} \tag{6}$$

is the modulus propagator acting on a $path\ p$.

To keep the geometric invariant, these descriptors are smoothed by a scaled low-pass filter ϕ_J giving the scattering coefficient.

$$S[p]x(\mathbf{u}) = U[p]x \star \phi_J(\mathbf{u}) \tag{7}$$

With the $path$ $p + \lambda = (\lambda_1, \ldots, \lambda_m, \lambda)$, we can combine eq. (5) and eq. (6) to give:

$$\tilde{W}U[p]x = \{S[p]x,\ U[p + \lambda]x\}_\lambda \tag{8}$$

We develop eq. (8) in the convolutional neural network. We define the *zero-th order* scattering coefficient is $S_{O0} = x$. It is different from the root definition with $S_{J0} = x \star \Phi_J x$. Scattering coefficient of *first-order* (S_{O1}) and *second-order* (S_{O2}) can be written, respectively:

$$S_{O1}x = \left\{S_{O0} \star \phi_J(\mathbf{u}),\ \left|x \star \psi_{\lambda_1}\right| \star \phi_J\left(\mathbf{u}\right)\right\} \tag{9}$$

$$S_{O2}x = \left\{S_{O1}x \star \phi_J(\mathbf{u}),\ \left|\left|x \star \psi_{\lambda_1}\right| \star \psi_{\lambda_2}\right| \star \phi_J\left(\mathbf{u}\right)\right\} \tag{10}$$

The eq. (9) and eq. (10) specification is compatible with the deep neural architecture. It is traversed deep-first to optimize memory requirements and integrated easily with an end-to-end trainable pipeline.

Propose the Reverse Engineering Diagram: Scattering Transform is a data-independent representation and has various good properties. Some of them are the invariant stability to additive noise, shift, and small deformation.

We propose the novel idea to discover the best features of Scattering Transform. Our method is an associate of the convolutional neural network and Scattering Transform. We establish the learning system from scattering coefficients. The detailed diagram is described as Fig. 1.

The best model is designed in the next section. The training process is constrained both the outputs of our model and the scattering coefficients. The loss function is the measure of the square L2 norm between two outputs. The network is optimized with the Adam algorithm. The initial learning rate is 0.001, weight decay is 0.01, the batch size is 50. The model

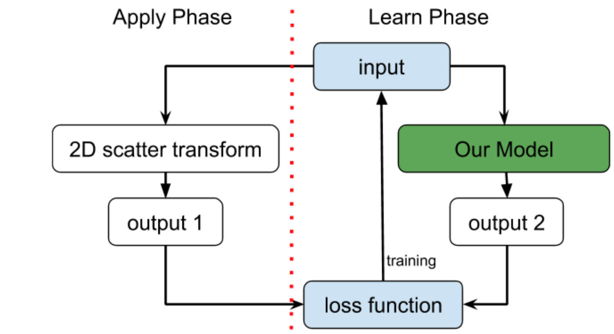

Fig. 1 *The reverse engineering with Scattering Transform*

The input is transformed by Scattering Transform and our model. The corresponding results are output1 and output2. The training process will optimize the loss function of the two output.

is trained until the loss function converges to zero value or reached the limit of epoch number.

To comprehend for objectivity from data, we test on much different data source. The first dataset is created randomly with 1000 samples of size [512,128]. Besides, we also examine the well-known datasets as CIFAR-10, Tiny ImageNet. In the training process, we also compare the results between our reliable model and the regular neural convolutional network.

Design the Complex Modulus Convolutional Neural Network Architecture: The model design idea based on the complex wavelet components, the modulus non-linear operator, and the low-pass filter ϕ in Scattering Transform theory. We design the model as described in formulas (9) and (10)

The complex wavelet ψ has the *real* and *imaginary* parts. They are implemented by two convolutional neural layers separately, which are the real and imaginary convolutional layers. The outputs of the real and imaginary convolutions are combined by the complex modulus non-linear operator. This result is then smoothed by the low-pass filter.

The low-pass filter is also presented by a convolutional neural layer. The result of this layer is concatenated with the output of the complex wavelet component across the channel dimension. Fig. 2 describes the complex modulus CNN architecture that corresponds to the *first-order* scattering representation.

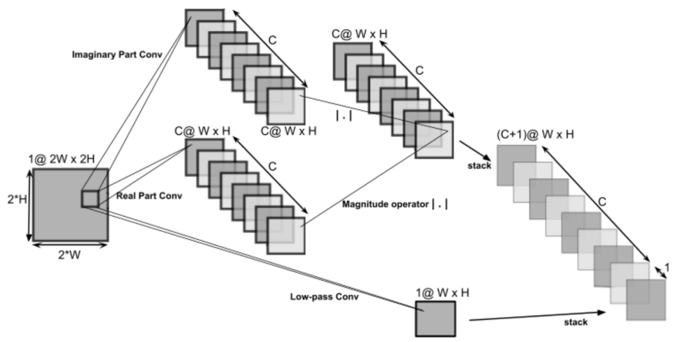

Fig. 2 *Complex Module Convolutional Neural Network Architecture*

This figure corresponds to the first-order Scattering Transform with L = 8. Its description as in eq. (9). It has 9 output channels and is downsampled.

The code is implemented the *first-order* Scattering Transform in PyTorch below. With L is number of angles in Scattering Transform; C is the input channels, K is the kernel size, S is the stride, P is the padding amount in a convolution layer.

```
class Scatt_OneOrder(nn.Module):
 def __init__(self, L, C, S):
   super(Scatt_OneOrder, self).__init__()
   self.L = L
   self.C = C
   self.phi = nn.Conv2d(C,C,K,S, P, bias=False, groups=C)
   self.psi_real = nn.Conv2d(C, C * L, K, S, P, False, C)
   self.psi_imag = nn.Conv2d(C, C * L, K, S, P, False, C)
   self.pad = nn.ZeroPad2d(P)

 def forward(self, x):
   s0 = self.phi(x)
   u1 = torch.sqrt(self.psi_real(x)**2 + self.psi_imag(x)**2)
   u1_smooth = F.conv2d(self.pad(u1),
               self.phi.weight.data.repeat(self.L, 1, 1, 1),
               bias=None, groups = u1.size(1))

   s0_chunk = torch.chunk(s0, self.C, 1)
   u1_chunk = torch.chunk(u1_smooth, self.C, 1)

   result = []
   for i in range(self.channel_in):
       result.append(s0_chunk[i])
       result.append(u1_chunk[i])
   s1 = torch.cat(result, axis=1)
   return s1
```

Repeating the same process, we will achieve the second-order scattering coefficient, as in eq. (10).

Some noticed things in designing are the size of the kernel and subsample operator. We use the *stride* property of the convolutional layer instead of the *Pool* layer, and consider the dependence between the model performance and the size of the kernel. We also compare the loss function of our model and the regular convolutional neural model.

Results: In this section, we will show some results. We consider the efficiency of the model, choose the size of the kernel in the convolution layer, extract and measure the invariant stable filters, and compare the speed to the Kymatio.

The efficiency of the model is evaluated through the loss function. We trained our model for the random dataset, CIFAR-10, and Tiny ImageNet. After training for 30 epochs, the loss function is converged to zero value. But training on the same hyperparameters, the regular convolutional neural network can't be converted to zero value. Fig. 3 is the loss function results for the CIFAR-10 dataset.

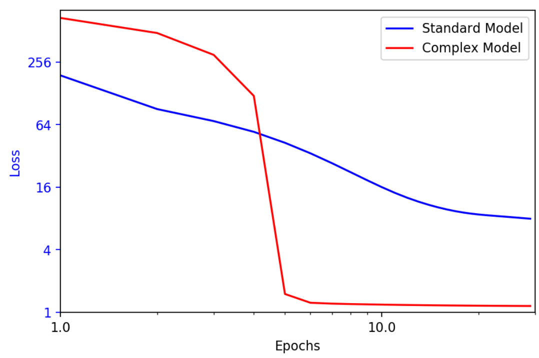

Fig. 3 *The loss function of the regular CNN and the complex modulus CNN for FIRAR-10.*

The regular CNN combines an ordered sequence of the convolutional layer, the Rectified Linear Unit layer, and the Pooling layer. The blue line is the loss function of the regular CNN. The red line is the loss function of our model, that is optimized.

The size of the kernel is also an important factor that make the model can be converge. The CNNs usually use in small kernel sizes. We set the size of kernel 3×3, 5×5, 7×7, and compare with each other. Fig.4 is the results of the loss function for the complex modulus CNN model that is the variable kernel size. We recognize the kernel of size 7×7 is the best choice.

We have trained the model on the dataset is created randomly 1000 samples of size 512×128. The number of angles L is set to 8, as the default in Scattering Transform. With the input of size $[1, 1, 512, 128]$, the output of the second-order model has size $[1, 81, 128, 32]$. Training for 60 epochs, we extract the weights of the model as described in Fig. 5.

For training on CIFAR-10 dataset, we use the test set with 10000 images of size $[32, 32]$. We select the green color channel for testing. With the input of size $[1, 1, 32, 32]$, the output of the second-order model is size $[1, 81, 8, 8]$. The weights of the model is extracted as showed in Fig. 6.

We have selected the test set on the Tiny ImageNet dataset for training. It has 10000 images of size $[64, 64]$. We only test on the green color channel. With the input of size $[1, 1, 64, 64]$, the output of the second-order model is size $[1, 83, 16, 16]$. Fig. 7 shows the filters extracted for the trained model.

Observing Fig. 5, 6, and 7, we have recognized the same patterns. They are very similar to the ripple-like of Morlet Wavelet.

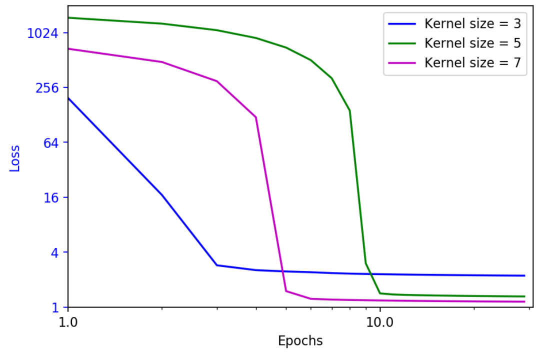

Fig. 4 *The loss function of the complex modulus model with the different kernel sizes.*

The blue, green, ping lines are the loss functions of the model with a kernel of size 3×3, 5×5, 7×7 respectively. The ping line is the lowest value which is converted to zero.

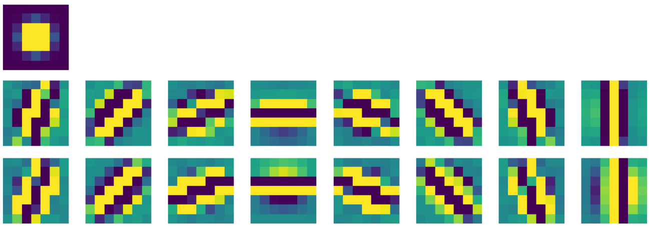

Fig. 5 *The geometric stable filter group is extracted when training for the random dataset.*

Row 1 is the low-pass filter of size 7×7. Rows 2 and 3 are eight real and imaginary filters of size 7×7 on each row, respectively.

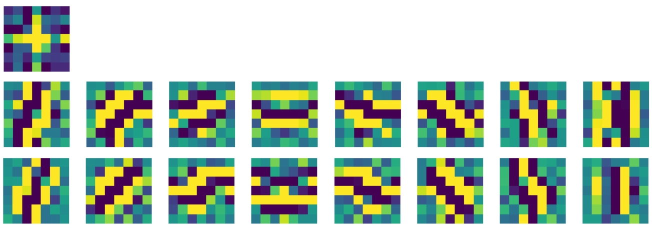

Fig. 6 *The filter group is extracted when training the model on CIFAR-10 dataset.*

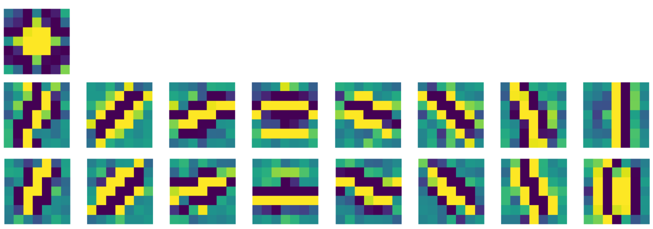

Fig. 7 *The filter group is extracted when training the model on Tiny ImageNet dataset.*

In this section, we measure the invariant stability of our method and the second-order Scattering representation. We test three desirable properties and compare the speed to the Kymatio framework. They are stability to additive noise, stability to shifts, and stability to deformation.

We take the first 2048 samples (called is x) in validation set from Tiny ImageNet dataset. A noise additive representation is $y(u) = x(u) + \epsilon(u)$, with $\epsilon(u)$ is the Gaussian noise. A shift representation $y(u) = x(u - c)$, with a shift c of 2 pixels right and 2 pixels down. A small deformation representation $x_\tau(u) = x(u - \tau(u))$, with τ is the random deformation grid of standard deviation $\sigma = 3$. We apply Scattering Transforms and our model for (x, y), we archive (S_x, S_y) and (V_x, V_y), respectively. The size of x, y is $[2048, 3, 64, 64]$ and the size of the outputs S_x, S_y, V_x, V_y is $[2048, 243, 16, 16]$. We calculate the L2 norm of each pair of value $(x, y), (S_x, S_y), (V_x, V_y)$. The results are showed in Table 1, our method has better stability than Kymatio every feature.

Table 1: Measure the invariant stability of noise, shift and deformation

Test	$\|\mathbf{x} - \mathbf{y}\|^2$	Kymatio $\|\mathbf{Sx} - \mathbf{Sy}\|^2$	Our method $\|\mathbf{Vx} - \mathbf{Vy}\|^2$
Addition noise	0.490209549	0.001616994	0.000192489
Shifts	0.725464642	0.002832161	0.001696022
Deformation	0.637948691	0.002723926	0.002358358

We compare the speed of first-order and second-order Scattering Transform in Kymatio to our method. The benchmark data is the CIFAR-10 dataset. We test on the CPU and GPU. Table 2 shows the execution times in seconds (averaged over each epoch) measured on a machine with a CPU Intel 8600K, GPU GTX1060, RAM 64G. Our method significantly outperforms the speed of Kymatio.

Table 2: Speed of Scattering Transform and Our method

Order	CPU		GPU	
	Scatter	Our method	Scatter	Our method
1^{st} order	37.5701	0.5909	0.2015	0.0037
2^{nd} order	180.37	2.22	4.65	0.023

Conclusion: Our outstanding work has proposed the training diagram for extracting the geometric stable filter group and design the complex modulus convolutional neural block. This filter group achieved fully the properties from Scattering Transform. The combination of the invariant stable filter group and the weight initialization for the convolutional neural layers leads the powerful solution to speed up Scattering Transform in any deep learning frameworks. These works are implemented completely in neural network, which is easily integrated into the end-to-end trainable system. While the pass works based on Fourier Transform leads to some limit for speed, flexibility, backend support in deep learning.

The complex modulus convolutional neural architecture is a potential candidate for competing for performance with some neural blocks in RestNet, MobileNet, GoogleNet. Its design is well understood based on analyzing the complex wavelet and non-linear operator. It is energy-preserving with the finite depth. The further work, we continue to discover the complex modulus neural architecture to design the new efficient models in deep learning. It is also required to develop the upper and lower bound tool to estimate the stability of each feature.

Acknowledgment: This work has been supported by The IET

J. Smith and A. N. Other (*The IET, Stevenage, UK*)

E-mail: jsmith@theiet.org

References

1 J. Bruna and S. Mallat. Invariant scattering convolution networks. *IEEE Trans. Pattern Anal. Mach. Intell.*, **35(8)**:1872–1886, 2013. doi: 10.1109/TPAMI.2012.230.

2 S. Mallat. Group Invariant Scattering. *Communications on Pure and Applied Mathematics*, **65**:1331-1398, 2012. doi: 10.1002/cpa.21413

3 Eickenberg, Michael Exarchakis, Georgios Hirn, Matthew Mallat, Stéphane. Solid Harmonic Wavelet Scattering: Predicting Quantum Molecular Energy from Invariant Descriptors of 3D Electronic Densities. *NeurIPS*, 2017.

4 F Gama, J Bruna, A Ribeiro. Stability of Graph Scattering Transforms. *NeurIPS*, 2019. arXiv:1905.04497

5 E. Oyallon et al. Scattering networks for hybrid representation learning. *IEEE Trans. Pattern Anal. Mach. Intell.*, **41(9)**:2208–2221, 2018. doi: 10.1109/TPAMI.2018.2855738.

6 Fergal Cotter , Nick Kingsbury. A Learnable ScatterNet: Locally Invariant Convolutional Layers. *ICIP*, **350-354**, 2019. doi: 10.1109/ICIP.2019.8802977.

7 Mathieu Andreux, Tomás Angles, Georgios Exarchakis, Roberto Leonarduzzi, Gaspar Rochette, Louis Thiry, John Zarka, Stéphane Mallat, Joakim Andén, Eugene Belilovsky, Joan Bruna, Vincent Lostanlen, Muawiz Chaudhary, Matthew J. Hirn, Edouard Oyallon, Sixin Zhang, Carmine Cella, Michael Eickenberg. Kymatio: Scattering Transforms in Python. *Journal of Machine Learning Research*, **21(60)**:1−6, 2020. url: http://jmlr.org/papers/v21/19-047.html.

8 Fergal Cotter, Sean McLaughlin. 2D Wavelet Transforms in Pytorch, 2019. doi: 10.5281/zenodo.3471950.