# TSViz: Demystification of Deep Learning Models for Time-Series Analysis

**Shoaib Ahmed Siddiqui**[1,2], **Dominik Mercier**[1,2], **Mohsin Munir**[1,2], **Andreas Dengel**[1,2], **Sheraz Ahmed**[1]
[1] German Research Center for Artificial Intelligence (DFKI GmbH), Kaiserslautern, Germany.
[2] TU Kaiserslautern, Kaiserslautern, Germany.
firstname.lastname@dfki.de



## Abstract

This paper presents a novel framework for demystification of convolutional deep learning models for time series analysis. This is a step towards making informed/explainable decisions in the domain of time series, powered by deep learning. There have been numerous efforts to increase the interpretability of image-centric deep neural network models, where the learned features are more intuitive to visualize. Visualization in time-series is much more complicated as there is no direct interpretation of the filters and inputs as compared to image modality. In addition, little or no concentration has been devoted for the development of such tools in the domain of time-series in the past. The visualization engine of the presented framework provides possibilities to explore and analyze a network from different dimensions at four different levels of abstraction. This enables the user to uncover different aspects of the model which includes important filters, filter clusters, and input saliency maps. These representations allow to understand the network features so that the acceptability of deep networks for time-series data can be enhanced. This is extremely important in domains like finance, industry 4.0, self-driving cars, health-care, counter-terrorism etc., where reasons for reaching a particular prediction are equally important as the prediction itself. The framework [1] can also aid in discovery of the filters which are contributing nothing to the final prediction, hence, can be pruned without any significant loss in performance.

## 1 Introduction

Despite of astonishing results from deep learning based models in a range of applications which includes computer vision, speech analysis, translation systems etc. [Krizhevsky *et al.*, 2012; Dahl *et al.*, 2010; Wu *et al.*, 2016], there has been limited applicability of these high performance models due to their black-box nature and lack of explainability of their decisions. This is specifically applicable in domains like business, finance, natural disaster management, health-care, self-driving cars, industry 4.0 and counter-terrorism where reasons of reaching a particular decision are equally important as the prediction itself [Knight, 2017].

There have been significant attempts to uncover the black-box nature of these deep learning based models [Yosinski *et al.*, 2015; Zeiler and Fergus, 2013; Simonyan *et al.*, 2013; Kumar *et al.*, 2017; Tishby and Zaslavsky, 2015; Zhang *et al.*, 2016], where visualization of model has been the most common strategy. Almost all of the proposed visualization systems are image centric due to ease of visualization and interpretability for humans in image space. Most of the ideas are equally applicable to time-series, but their unintuitive nature makes it difficult to directly transfer these ideas for improved human understanding of the model.

This paper presents a novel framework for demystification of deep models for time-series analysis. In particular, the contribution of this paper is threefold:

- A novel 3D visualization framework for time-series deep learning models. This framework is generic and is capable of visualizing any convolutional deep learning model for time-series analysis. We believe, that the presented framework will aid in making informed and explainable decisions.
- An influence tracing algorithm to discover the important filters within each layer. The influence and importance of a particular filter is obtained by leveraging the back-propagation algorithm. Filter importance is computed based on its influence on the final output. Similarly, input saliency map is obtained by computing the influence of the input on the current/selected filter outputs. This provides a detailed functional view of the deep learning model.
- A clustering based approach to understand the range of different types of filters present within a particular layer of the network. Filters belonging to the same cluster are similar in terms of their activation pattern. This provides a good estimate regarding the diversity present in the network.

The presented visualization tool provides an opportunity to explore the network at four different levels of abstraction i.e., from abstract to detailed view. First level provides an overview of network architecture which includes number of

[1] Framework download link: https://hidden.for.blind.review

convolutional, dense and max-pooling layers. On the second level, it computes and provides an overview of most influential/important filters in each layer, utilizing the framework of backpropagation. The third level provides the possibility to visualize all the filters of a selected layer. The importance of the filter along with the influence of the input in the computation of its output is visible to the user at all levels. The final level refines the presented information further by providing a view where similar filters are clustered together to provide an estimate regarding the diversity obtained by the network during training.

The presented framework provides the possibility to demystify the deep networks for time-series analysis by providing meaningful explanation for the predictions/classifications. Furthermore, with the understanding and identification of the important items in the network, it can be pruned to improve generalization capabilities without explicit reliance on strong regularization techniques.

## 2 Related Work

Significant efforts have been made to understand the learnt features of a model in order to demystify the deep learning black-box. This is specifically the case for visual recognition models since visualizing the kernels themselves and their activations can give hints about the feature that the system is learning and the kernels themselves are directly interpretable by humans [Yosinski *et al.*, 2015; Zeiler and Fergus, 2013; Simonyan *et al.*, 2013]. One of the most common and effective technique for visualization of network learning on visual data is saliency maps [Bojarski *et al.*, 2017] which highlights the regions which the network focused on in order to generate a particular prediction. In order to visualize human understandable features that the neuron is responding to, there have been significant efforts in detecting which input maximally activates a neuron [Girshick *et al.*, 2013]. The problem of discovering the part of input that was most responsible for a particular prediction has also been extensively studied [Yosinski *et al.*, 2015; Zeiler and Fergus, 2013; Simonyan *et al.*, 2013]. [Shrikumar *et al.*, 2017] presented an alternate formulation from gradient based methods for obtaining the maximally activating input. [Li *et al.*, 2016] used saliency maps to identify focus regions for textual data.

Despite of these advancements, the area of network visualizations for time-series analysis has remained unexplored till now. A recent attempt has been made by [Kumar *et al.*, 2017] to visualize the input points which were most influential for a particular prediction through gradients.

Theoretical contributions have also been made in order to understand the amazing generalization capabilities of these deep models. [Zhang *et al.*, 2016] presented empirical analysis to divert attention to the philosophical topic of what actually is perceived as generalization. Information bottleneck theory [Tishby and Zaslavsky, 2015] has also gained popularity as method for explaining the generalization and learning capabilities of deep learning based models. [Koh and Liang, 2017] presented influence functions as a methodology to trace back model predictions in terms of its training data. This analysis enabled discovery of dataset errors, model debugging, and creation of visually-indistinguishable adversarial training examples which are able to flip the network's test time predictions.

We refer the readers to [Montavon *et al.*, 2017] for a comprehensive review of the previous work in this direction based on the tutorial given at ICASSP (2017).

## 3 TSViz: The Proposed Approach

TSViz is based on the principle of backpropagation proposed by [Rumelhart *et al.*, 1986], which is essentially the chain rule from calculus. Backpropagation algorithm provides an efficient way to compute influences of the tensor w.r.t. another tensor, where the tensor can be in $\mathbb{R}^{n \times n}$. The same framework is the workhorse for training of the deep learning models where the influence of the network parameters is computed on the final cost function. We leverage this capability to compute influences for uncovering the deep learning black-box by computing the influences of the input on the current filter which is the input saliency map. This also enables the discovery of the filter's importance by computing its influence on the final prediction of the network.

TSViz provides possibility to visualize any convolutional network from several dimensions, on different level of abstraction, starting from an abstract representation of whole network, including layer-wise view, important filters on each layer (Section 3.2), as well as grouping of filters which are similar exhibiting same/similar trend (Section 3.3). Furthermore, these clustered filters can also be visualized on detailed level to understand similarities and difference within and between the clusters.

### 3.1 Backpropagation

As TSViz framework is based on backpropagation, this section provides the basic concept of the backpropagation algorithm. The basic aim of learning step in neural network is to reduce the cost function $C$. A cost function computes the difference between the network and the desired output. Ideally, the network must output a value which is same as target. The whole learning step is about reducing the discrepancy between the two values. As the network output is calculated based on the weights and biases of different neurons involved, these weights and biases are adapted during the learning process in order to reduce the cost function. Ultimately, backpropagation is about understanding how the change in the weights and biases of a network affect it's cost function. These changes can be obtained by calculating the partial derivatives of cost function with respect to any weight $w$ or bias $b$ as $\partial C/\partial w$ and $\partial C/\partial b$.

Backpropagation algorithm can be decomposed into four steps which includes: (i) Feed-forward pass through the network, (ii) Backpropagation to the output layer, (iii) Backpropagation to the hidden layers, (iv) Update network parameters. In the feed-forward pass through the network, the output of all the hidden neurons and the final network output is computed based on the randomly initialized weights along with the evaluation of the final cost function. The networks in deep learning are mostly comprised of both convolution and fully connected layers. Rectified Linear Unit (ReLU) is used as

the non-linearity/activation function. The activation for fully connected and convolutional layers is presented in Eq. 2 and Eq. 4 respectively. $a_j^l$ denotes the activation of the j[th] neuron in the l[th] layer (for fully-connected layers) while $a_{ji}^l$ denotes the activation of the j[th] neuron in the l[th] layer at the i[th] input location (for convolutional layers). $k$ is defined as $\lfloor FilterSize/2 \rfloor$ for convolutional layers while $e$ denotes the number of neurons in the last layer $l-1$.

$$z_j^l = \sum_{k=1}^{e} W_{jk}^l a_k^{l-1} + b_j^l \tag{1}$$

$$a_j^l = max\left(z_j^l, 0\right) \tag{2}$$

$$z_{ji}^l = \sum_{-k}^{k} W_{jk}^l a_{i-k}^{l-1} + b_j^l \tag{3}$$

$$a_{ji}^l = max\left(z_{ji}^l, 0\right) \tag{4}$$

The error is backpropagated to the initial layers, and the gradient with respect to the network parameters is computed (weights and biases). The error ($\delta$) of $j^{th}$ neuron at the output layer $L$ is presented in Eq. 6.

$$\delta_j^L = \frac{\partial C}{\partial a_j^L} \frac{\partial a_j^L}{\partial z_j^L} \tag{5}$$

$$\delta_j^L = \frac{\partial C}{\partial a_j^L} max'\left(z_j^L, 0\right) \tag{6}$$

$$\delta_j^l = \left(\left(w_{ij}^l\right)^T \delta_j^{l+1}\right) \odot max'\left(z_j^l, 0\right) \tag{7}$$

$$max'(x,0) = \begin{cases} 1, & \text{if } x > 0 \\ 0, & \text{otherwise} \end{cases} \tag{8}$$

Once gradient of the error w.r.t. output layer is computed, the error is backpropagated to all neurons in the hidden layers using Eq. 7. The gradient of ReLU is 1 where the value of input $x$ exceeds 0 and remains 0 otherwise as mentioned in Eq. 8. Similarly, the max-pooling layer has gradient equal to 1 wherever the maximum quantity occurred and remains 0 otherwise. The rate of change of cost w.r.t. the bias and weights in the l[th] layer is given in Eq. 9 and Eq. 10 respectively.

$$\frac{\partial C}{\partial b_j^l} = \delta_j^l \tag{9}$$

$$\frac{\partial C}{\partial w_{jk}^l} = a_k^{l-1} \delta_j^l \tag{10}$$

After computation of the gradients, the network parameters (weights and biases) are updated in the negative gradient direction.

### 3.2 Influence Computation

For TSViz, we derive our influence computation strategies leveraging the framework of backpropagation explained in Section 3.1. Instead of computing the gradient of the layer w.r.t. the loss/cost function, we compute the gradient of the output layer w.r.t. the current layer $d$. This provides us with a point-wise estimate about how each value impacts the output activation $a^L$. Since both positive and negative influences are equally important to us, we consider the 1p-norm of the gradient. Furthermore, as we are interested in the overall influence of a particular filter, rather than point-wise influence, the values of the influences can be reduced to a single value by summation. This operation retains the information as the sum of the influences of each of its components provides a good estimate regarding the importance of the overall filter. Eq. 12 provides the mathematical formulation of the influence of layer $d$ on the final output $a^L$.

$$\delta_j^l = \frac{\partial a^L}{\partial a_j^l} \frac{\partial a_j^l}{\partial z_j^l} \tag{11}$$

$$I_{output}^d = \sum_j \left|\delta_j^d\right| \tag{12}$$

Another interesting influence originates from the input, which is computed and visualized as saliency map. This information provides important insights regarding the data points in the input that the filter is actually responding to for computation of its output. This value can similarly be obtained by computing the gradient of the current layer w.r.t. the input layer. Since we need point-wise estimates, there is no need for any reduction. We again use the 1p-norm of the gradient as the magnitude is of relevance irrespective of the direction. We denote the input as $a^0$, therefore, this influence of the input can be computed using Eq. 15.

$$\delta_j^l = \frac{\partial a^d}{\partial a_j^l} \frac{\partial a_j^l}{\partial z_j^l} \tag{13}$$

$$\delta_j^0 = \frac{\partial a^d}{\partial a_j^0} \tag{14}$$

$$I_{input}^d = \left|\delta_j^0\right| \tag{15}$$

In order to be able to visualize and compare the activation intensities of the different filters within a particular layer, the absolute values of the influences are scaled using the min-max scaling presented in Eq. 16.

$$I_{output}^l = \frac{I_{output}^l - \min_k I_k^l}{\max_k I_k^l - \min_k I_k^l} \tag{16}$$

In order to avoid information overload for the user, a percentile filter has been integrated into TSViz. This percentile filter enables the user to only visualize the filters which were highly influential for the generation of the final prediction. Figure 7 provides an example application of the percentile filter onto the second level view of the network.

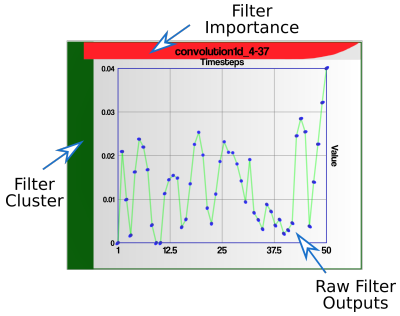



Figure 1: Detailed view of a particular filter

### 3.3 Filter Clustering

While visualizing different filters on each layer, we realized significant amount of redundancy between filters of a particular layer. To better understand different trends available in each layer, it is important to cluster similar/redundant filters. Also, in order to reduce the amount of information that needs to be processed by the user, it is essential to have information regarding which filters are essentially similar to other filters. Therefore, we propose a clustering based approach for clustering different filters with similar activating patterns.

We used K-Means++ clustering algorithm to cluster filters on the basis of their similarity in the activation pattern using Euclidean distance as the distance metric. We encode each filter via its activation vector $x \in \mathbb{R}^D$ where $D$ denotes the number of output neurons within that particular layer. The filter cluster centers are represented by $C \in \mathbb{R}^{k \times D}$ where $k$ denotes the number of clusters. Let $z$ be equal to $k$ randomly initialized whole numbers between 1 and $n$, then the initialization of the cluster centers can be formulated as Eq. 17. The distance of the data points $X \in \mathbb{R}^{n \times D}$ from the cluster centers is then computed using Eq. 18.

$$C_i = X_i \quad \forall i \in z \tag{17}$$

$$y_i = \arg\min_{j=1,\ldots,k} \|X_i - C_j\| \tag{18}$$

The assigned cluster centers are then used to updated the cluster centers as described in Eq. 19.

$$C_i = \frac{1}{|y_i|} \sum_{X_i \in y_i} X_i \tag{19}$$

The cluster centers are iteratively updated using Eq. 18 and Eq. 19 until the cluster centers converge. The number of clusters $k$ is defined to be one-fifth of the number of filters $n$ in that particular layer. In order to achieve clustering based on just the activating pattern rather than the magnitude, we again scale the raw activation values $X$ using min-max scaling as defined in Eq. 16. This makes the clustering invariant to the activation scale.

## 4 Implementation

We used Keras with TensorFlow backend [Chollet and others, 2015; Abadi, 2015] developed specifically for deep learning. TensorFlow is an automatic differentiation package which enables automatic computation of the gradients. We leveraged this capability to compute the influences as described in section 3. As a case-study, we trained a network on Internet Traffic Dataset [Cortez *et al.*, 2012] for time-series forecasting (we only used the B5M dataset out of the six available variants). The network operates on the input size of 50 time-stamps and is comprised of three convolution, two max-pooling and one dense layer. The network is trained using Adam optimizer with a learning rate of 0.001 for 5000 epochs using a batch size of 5 and mean squared error as the loss function. This network was able to achieve state-of-the-art forecasting results on the aforementioned dataset. The network takes in a bi-channel input where the first channel corresponds to the original input signal while the second channel is comprised of the first-order derivative of the original signal.

The front-end for the visualization toolbox is developed using Unity Game Engine [Technologies, 2005] which communicates with the backend exposed as a RESTful API. This decouples the model from the visualization aspect. Even though we choose a forecasting model (which differs significantly from the models we usually encounter in the image domain), the system is generally applicable to any deep learning model for time-series analysis as the it is only dependent on the effective computation of gradients.

## 5 Analysis and Discussion

This section is explicitly focused on providing detailed insights regarding different dimensions from which a network can be explored. In particular, the TSViz toolbox provides possibility to explore and analyze the given network at four different levels of abstraction.

The first level provides the most abstract representation of the complete network. This provides the user with the opportunity to get himself acquainted with the network in question by providing a simplistic view of the network architecture highlighting the number of convolutional, max-pooling and dense layers present in the network along with the original input signal in question. The user can switch to the detailed view through the defined shortcut keys. The visualization is created in 3D so the user can change his perspective, but the frontal view is the most appropriate for understanding due to its simplicity. Figure 2 provides an overview of the complete network architecture both visually and in textual form.

The second level extends the network architecture overview to display the original filters instead of dummy-blocks. This view is designed to get an overall idea regarding

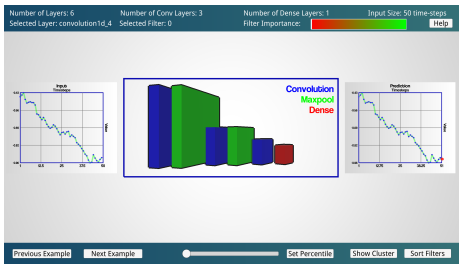



Figure 2: Abstract representation of the network (First Level)

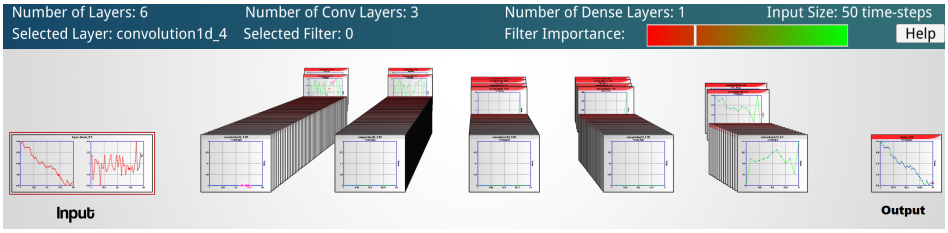



Figure 3: Filters sorted according to their importance (Second level)

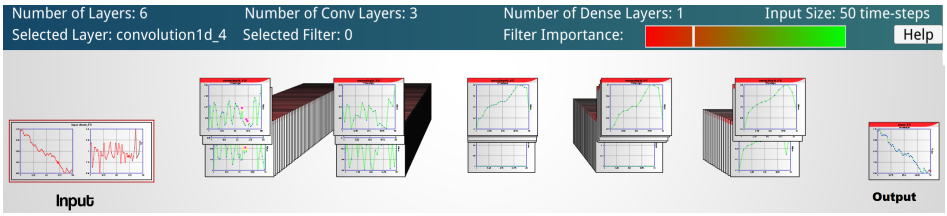



Figure 4: Filters sorted according to their importance in descending order (Second level)

the learning of the network. We leverage conventional line plots which is one of the best known techniques for this purpose. Figure 3 and Figure 4 provides an unfolded view of the network. The plots in a particular layer are stacked in the depth dimension which the user can scroll through using the mouse-wheel to get an idea about the different filters present in a particular layer. As the user scrolls through the different filters within a layer, the input automatically adapts to demonstrate the influence of the input on the particular layer in question. The color strength of the red bar (Figure 1) on top of each filter indicates its impact on the final output i.e. the filter's importance. The filters with higher importance pops out of the stack with the height being proportional to its importance. The user has the possibility to sort the filters in ascending or descending order w.r.t. their importance as presented in Figure 3 and Figure 4 respectively.

The third level focuses on a particular layer instead of the whole network. This view displays all the filters in a particular layer stacked on a 2D grid in order to be able to inspect the layer in detail. This view again supports sorting w.r.t. to the filter importance within a particular layer. Just like the first view, the third view retains the input and output plots in order to make sure that the user is equipped with all the important information for analysis. Figure 5(a) provides a depiction of the third level.

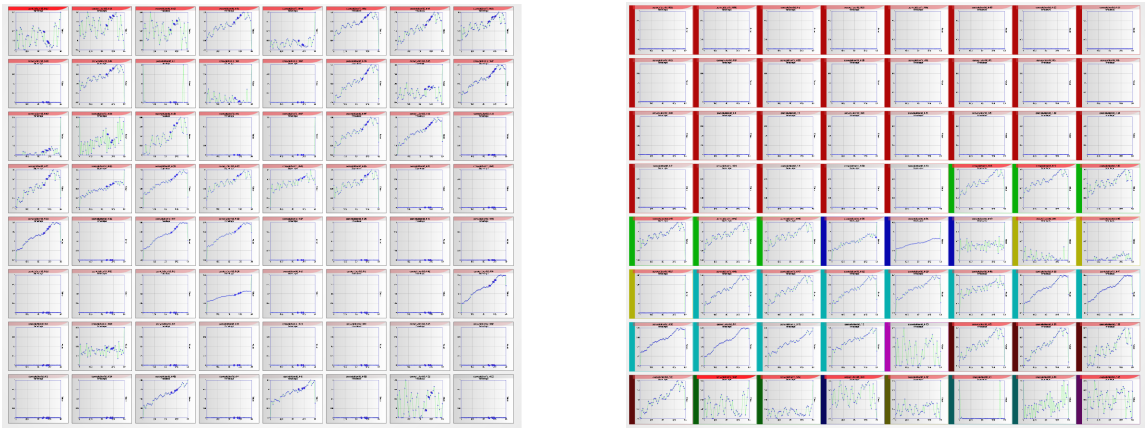
The fourth level further enhances the information from the third level by additionally visualizing the information regarding the filter clusters. Filters with similar activating patterns belongs to the same cluster. The clusters are represented via a colored bar on the left side of the plot. Filters belonging to the same cluster has the same color bar. Figure 5(b) provides a depiction of the fourth level, where it can be clearly seen that all the filters in first cluster have no activity for that particular input. These filters amounts to a large fraction of the total filters present in the layer. Similarly, other clusters discovered by the system demonstrates consistency in the trend disregarding the scale.

Ability to take a look at the items within the receptive field of a particular neuron which were responsible to produce that particular value is essential for in-depth understanding of the network. Therefore, the visualization framework is equipped with Recursive highlightion feature which highlights the points which are actually responsible for the activation value obtained at that particular point. Adding lines to demonstrate the influence was problematic as the scene becomes cluttered with useless items. In order to highlight the points which are responsible, we add a light emission property to these points along with increased diameter making them stand out from the rest of the values. This feature enables the user to understand the flow of information. The feature is highlighted in Figure 6. It is important to note that instead of visualizing all the values within the receptive field of the neuron, we only visualize the values which were actually responsible to produce the current activation, therefore, we only trace the maximum values through the max-pooling layers as the suppressed values have no impact onto the obtained activation.

Since a network can be comprised of a large number of filters, the system is equipped with a percentile constraint where the user can specify the importance percentile so that only the filters which exceed that importance percentile are visualized within each layer to allow the user to focus on the more important details. Figure 7 provides a glimpse of the percentile filter in action. In this case, the network is same as the one presented in Figure 3 and Figure 4 but only visualizes the filters which were most important due to the percentile filter being set to the $90^{th}$ percentile.

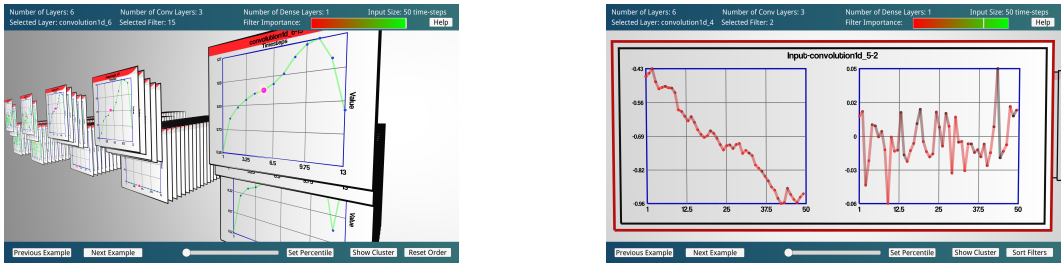
The visualization enabled a detailed inspection of the network which highlighted many different aspects of the network's learning.

- Most of the filters in the network were useless i.e. they contributed nothing to the final prediction for that particular input. These filters changed as the stimulus to the network changed indicating that some filters were specialized in a particular input.



(a) (b)

Figure 5: Filters stacked on 2D grid (Third level - left) along with information regarding the filter cluster (Fourth level - right). All the filters are sorted w.r.t. their importance.



(a) (b)

Figure 6: Recursive highlightion (left) and influence tracing of the input (right)

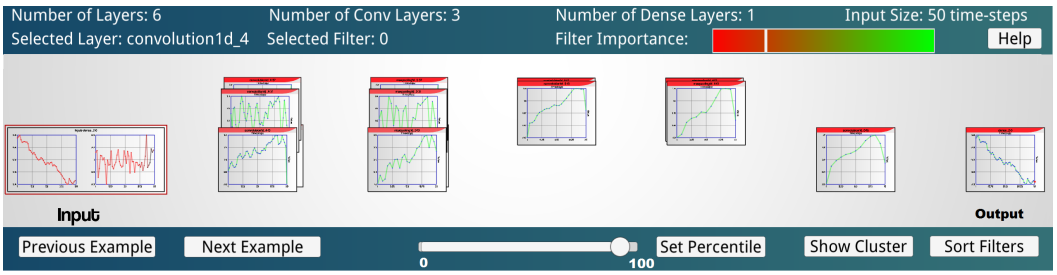



Figure 7: Application of the percentile filter on the detailed view (Second level)

- Many of the filters had very similar activating patterns in the network which were assigned to the same filter cluster. This highlighted the aspect of lack of diversity in the trained network.
- Despite of the improvement in performance with the addition of the first-order derivative of the original signal, most of the filters strongly attended to the original signal as compared to the first-order derivative.

## 6 Conclusion

We proposed a novel visualization framework for time-series based deep learning models. The framework enabled the understanding of the model as a parametric function. The different views available within the visualization enabled in-depth exploration of the network architecture. This exploration will help in understanding of the network itself as well as enable new improvements within the architecture by the insights gained through the visualization.

We are currently working on extending this framework to be utilized for network pruning based on the filter clusters and filter importance. The pruned network might demonstrate superior generalization capabilities without explicit reliance on strong regularization. This is possible as the parts of the network responding/tuned to highly specific stimulus might contribute to the overfitting of the network.

## References


[Abadi, 2015] Martín Abadi. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[Bojarski *et al.*, 2017] Mariusz Bojarski, Philip Yeres, Anna Choromanska, Krzysztof Choromanski, Bernhard Firner, Lawrence D. Jackel, and Urs Muller. Explaining how a deep neural network trained with end-to-end learning steers a car. *CoRR*, abs/1704.07911, 2017.

[Chollet and others, 2015] François Chollet et al. Keras, 2015.

[Cortez *et al.*, 2012] Paulo Cortez, Miguel Rio, Miguel Rocha, and Pedro Sousa. Multi-scale internet traffic forecasting using neural networks and time series methods. *Expert Systems*, 29(2):143–155, 2012.

[Dahl *et al.*, 2010] George Dahl, Marc'aurelio Ranzato, Abdel rahman Mohamed, and Geoffrey E Hinton. Phone recognition with the mean-covariance restricted boltzmann machine. In J. D. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R. S. Zemel, and A. Culotta, editors, *Advances in Neural Information Processing Systems 23*, pages 469–477. Curran Associates, Inc., 2010.

[Girshick *et al.*, 2013] Ross B. Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. *CoRR*, abs/1311.2524, 2013.

[Knight, 2017] Will Knight. Mit technology review: The financial world wants to open ai's black boxes, 2017.

[Koh and Liang, 2017] P. W. Koh and P. Liang. Understanding black-box predictions via influence functions. In *International Conference on Machine Learning (ICML)*, 2017.

[Krizhevsky *et al.*, 2012] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[Kumar *et al.*, 2017] D. Kumar, G. W Taylor, and A. Wong. Opening the Black Box of Financial AI with CLEAR-Trade: A CLass-Enhanced Attentive Response Approach for Explaining and Visualizing Deep Learning-Driven Stock Market Prediction. *ArXiv e-prints*, September 2017.

[Li *et al.*, 2016] Jiwei Li, Will Monroe, and Dan Jurafsky. Understanding neural networks through representation erasure. *CoRR*, abs/1612.08220, 2016.

[Montavon *et al.*, 2017] Grégoire Montavon, Wojciech Samek, and Klaus-Robert Müller. Methods for interpreting and understanding deep neural networks. *CoRR*, abs/1706.07979, 2017.

[Rumelhart *et al.*, 1986] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, October 1986.

[Shrikumar *et al.*, 2017] Avanti Shrikumar, Peyton Greenside, and Anshul Kundaje. Learning important features through propagating activation differences. *CoRR*, abs/1704.02685, 2017.

[Simonyan *et al.*, 2013] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. *CoRR*, abs/1312.6034, 2013.

[Technologies, 2005] Unity Technologies. Unity game engine, 2005.

[Tishby and Zaslavsky, 2015] Naftali Tishby and Noga Zaslavsky. Deep learning and the information bottleneck principle. *CoRR*, abs/1503.02406, 2015.

[Wu *et al.*, 2016] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Łukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean. Google's neural machine translation system: Bridging the gap between human and machine translation. *CoRR*, abs/1609.08144, 2016.

[Yosinski *et al.*, 2015] Jason Yosinski, Jeff Clune, Anh Nguyen, Thomas Fuchs, and Hod Lipson. Understanding neural networks through deep visualization. In *Deep Learning Workshop, International Conference on Machine Learning (ICML)*, 2015.

[Zeiler and Fergus, 2013] Matthew D. Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. *CoRR*, abs/1311.2901, 2013.

[Zhang *et al.*, 2016] Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning requires rethinking generalization. *CoRR*, abs/1611.03530, 2016.