Locality Guided Neural Networks for Explainable Artificial Intelligence

Randy Tan, Naimul Khan, and Ling Guan
Department of Electrical and Computer Engineering
Ryerson University
Toronto, Canada
randy.tan@ryerson.ca, n77khan@ryerson.ca, lguan@ryerson.ca

***Abstract*—In current deep network architectures, deeper layers in networks tend to contain hundreds of independent neurons which makes it hard for humans to understand how they interact with each other. By organizing the neurons by correlation, humans can observe how clusters of neighbouring neurons interact with each other. In this paper, we propose a novel algorithm for back propagation, called Locality Guided Neural Network (LGNN) for training networks that preserves locality between neighbouring neurons within each layer of a deep network. Heavily motivated by Self-Organizing Map (SOM), the goal is to enforce a local topology on each layer of a deep network such that neighbouring neurons are highly correlated with each other. This method contributes to the domain of Explainable Artificial Intelligence (XAI), which aims to alleviate the black-box nature of current AI methods and make them understandable by humans. Our method aims to achieve XAI in deep learning without changing the structure of current models nor requiring any post processing. This paper focuses on Convolutional Neural Networks (CNNs), but can theoretically be applied to any type of deep learning architecture. In our experiments, we train various VGG and Wide ResNet (WRN) networks for image classification on CIFAR100. In depth analyses presenting both qualitative and quantitative results demonstrate that our method is capable of enforcing a topology on each layer while achieving a small increase in classification accuracy.**

I. Introduction

There has been significant progress in deep learning in the past few years, especially with recent advances in computational power and the emergence of large datasets to train models. Deep learning techniques have taken the top position as state of the art in many domains such as image processing [1] and natural language processing [2]. The development of software libraries have simplified the process of training networks to the point where domain experts that only have a small amount of deep learning knowledge can build models. However, one major challenge for deep learning is its inherent black box nature, in which users have little control or knowledge on what type of features are learned in the hidden layers. This lack of knowledge can prohibit the application of deep learning in critical domains; where misclassifications have a high cost. Until we can explain how deep learning models have come up with a decision, it will be difficult to apply them in areas such as self-driving cars, medical diagnoses, and other critical domains [3]. Another application where it is important to understand what type of semantic information a model learns is in transfer learning [4]. Many of these methods take pretrained networks that are trained on large datasets like ImageNet [5] or Kinetics [6] and then utilize only the feature extraction portion of the model and feed it through a different classifier. In these types of methods, the classifier is forced to treat all of its input features as independent and unknown features due to the blackbox nature of deep learning. If domain experts could understand the features from deep learning methods, they may be able to apply their own knowledge and form handcrafted features that could improve the classifier.

One challenge in trying to understand neural networks is that the neurons within a given layer are independent of each other. In current deep learning structures, neurons are only connected to other neurons in the previous and next layers; none of the neurons within the same layer are connected to each other. This makes it difficult to find any inherent relationship between features without comparing all of the neurons in the same layer together. We propose a new algorithm for back propagation inspired by Self-Organizing Map (SOM), that can enforce a topology on each layer where neighbouring neurons learn similar concepts. While in this work we focus on convolutional neural networks with images, this method is theoretically applicable to other networks types. Enforcing a topological structure that gathers similar filters together will make it easier for humans to visually understand the deeper layers of a network. The work by Google [7] identified the concept of *Neuron Groups* in which certain visual concepts activate specific groups of neurons. Our method aims to group these neurons together in the topology. The primary contributions of our Locality Guided Neural Network (LGNN) method include:

- Clustering neurons in each layer of a network using a similar neighbourhood function to SOM such that neighbouring filters share semantic concepts.
- Only modifying the gradient update step for convolutional layers such that it can be integrated into any state of the art Convolutional Neural Network (CNN) model with negligible computational overhead.
- Enforcing the topology during training without any post-processing, unlike methods like [8], [9]
- Slightly increasing accuracy for image classification due to the regularizing effect of neighbourhood functions

II. Background

A. Explainable Artificial Intelligence

The Explainable Artificial Intelligence (XAI) program was started by DARPA in their interest for autonomous systems that can make critical decisions. While they credit machine learning methods for their high success, they state their main limitation is their inability to explain their decisions to human users. The main questions they want answered by machine learning systems include: ”Why did you make your decision?”, ”When do you succeed or fail?”, and ”How can I correct your errors?”. The target for the XAI program is for a new generation of machine learning methods that can have their thought process explained to human users. Researchers have mainly targeted the problem of XAI in two ways:

- Post hoc methods that can attach explanations to already trained models
- Modifying or creating new machine learning models that are more inherently understandable by humans

Research in post hoc interpretation of machine learning have tried to tackle the problem from several angles. One example is Local Interpretable Model-agnostic Explanations (LIME) which ignores the AI model and explains the relationship between a single input and its output by approximating a small local area around the input as a linear classifier [10]. For XAI in CNNs, some recent works try to tie neuron activations to semantic concepts humans can understand. One method is Network Dissection [9], which is a framework that assigns quantitative values to how interpretable an individual neuron is to a semantic concepts such as scenes, objects, textures, and colors. Another method is Testing Concept Activation Vectors (TCAV) [8], which translates semantic concepts to a vector within a layer's output space rather than an individual neuron. TCAV finds the vector by first creating a training set consisting of images that contain a concept and images that do not. The images are fed through the network and the activations at the desired layer are used as an input to a linear classifier. The vector orthogonal to the linear boundary is used to represent the concept.

Several XAI methods, including our method, propose ways to modify models to be inherently more understandable. Variational Auto Encoders (VAE) and their variants are a good example of a specific network model that is explainable [11]–[13]. Autoencoders consist of an encoder that compresses an input to a latent space, and a decoder that restores the original input vector. VAEs modify the encoder such that it returns a probability distribution instead of a single latent variable and also adds a regularization term to the loss such that the latent space becomes continuous, where vectors close to each other in the latent space have similar appearances in the original input space. This allows users to sample the latent space and still get coherent decoded outputs. In one XAI application, [4] proposed a network architecture that inherently learns action recognition in an explainable way. Rather than classifying the action directly, their method detects objects first. The objects are then fed through a Graph Connected Network (GCNs) [14] to determine how the objects interact to classify actions. Since the network has been broken up into steps, a human can see what types of objects were detected by the first step, and also see what type of connections the second step deemed as important for classification.

Our method explores XAI by simply re-organizing the order of filters such that users can have a more global understanding of each layer compared to looking at the unordered filters individually. In the ideal scenario, the filters trained by LGNN would be identical to the baseline, but organized such that the filters that all activate on similar semantic concepts are gathered together. In [7], the authors identified the groups of filters that activated on similar concepts as *Neuron Groups*, but their work did not use these groups to reorganize the filters.

B. Network Visualization

Since CNNs are applied to the domain of image processing, it is natural that many methods try to explain CNNs through visualization. Two categories of visual CNN interpretations include attribution/saliency and feature visualization [15], [16]. Attribution methods show what parts of an example image triggered a specific neuron activation [7], [17]–[19]. They first take an input image and feed it through a trained CNN. They then try to map the neuron response back to the pixel regions in the original image that activated it. The issue with attribution methods is that they only show a correlation between an input image and neuron activation. In traditional CNNs, individual neurons at the higher levels can represent mixtures of patterns or concepts [20]. Since attributions are tied to dataset examples, they could possibly mislead users by only showing a portion of what a neuron is looking for.

Feature visualization methods try to numerically generate a new image that maximizes a specific neuron's activation. They start with a noise image and trains the image to maximally activates a neuron. It accomplishes this by setting the training loss to the negative activation of the desired neuron, freezing the network weights, and passing the gradients all the way to the noise image. The resulting image should be a pattern that represents what the desired neuron is looking for. There are several regularization techniques listed by [16] that can help make clearer visualizations including; high frequency penalization, transformation robustness, and learned priors.

C. Self Organizing Maps

This work uses a locality enforcing algorithm that is motivated by SOM [21]. SOM is an unsupervised clustering algorithm that is often compared to Vector Quantization (VQ) or K-Nearest Neighbours (KNN). The unique feature of SOM compared to other clustering methods is that SOM sorts its clusters by similarity. Section III-A explains in fuller detail, but essentially SOM achieves the locality based ordering by defining a structure (typically a 2-D grid) for the clusters during initialization as a set of connected neighbours for each node and then propagating any gradients that each node receives to its neighbours.

Several works including [22], [12] and [23] are examples of methods that have employed SOM in AI. In [22], Spherical Self Organizing Maps (SSOM) is used as an extension of SOM, to evaluate ballet dance performance. The only difference between SOM and SSOM is that SSOM is structured as a tessellated enclosed sphere as opposed to a 2D grid. This work treated individual neurons within the SSOM as a posture and classified full ballet performances as a trajectory through the SSOM. In [12], the main goal of the work was to take the high dimensional features from deep networks and to make them more visually interpretable for humans; specifically for time series. The work used a VAE as the baseline model and further compresses the encoding by representing it as a neuron within a trained SOM. The main contribution in the paper was the introduction of several loss functions, which allow the backpropagation to overcome the non-differentiability of discretization and for the SOM to be traversed smoothly in time. Similar to [22], this method can encode a time series as a trajectory through a SOM. In [23], a SOM was trained on the FC layer at the end of a deep network as a means of quantization for Aproximate Nearest Neighbour (ANN) search. Their method leveraged the topology preserving capabilities of SOM to train a quantization loss along side the classification loss where similar image pairs minimized their distance on the SOM map while dissimilar images maximized their distance. By attaching SOM to the end of a deep network, they were able to acquire input features from the FC layers that contain information about the image. Overall, their method is capable of both ANN search and classification.

III. Proposed Method

A. Locality Guided Neural Networks

The main goal of this work is to interfere as little as possible with current CNN structures while allowing correlation along the channel dimension to be inherent. Our method will attempt to enforce a topology on each layer such that neurons that search for similar concepts are clustered together. This will make it easier for users to have a more global understanding of each layer rather than looking at individual neurons. Additionally, having locality within the learning will allow information to be propagated within each layer instead of only propagating between layers.

Our algorithm borrows its topology preserving properties from SOM. In the original SOM algorithm, neurons are arranged into a 2D grid and contain a location in the grid l_j and a weight w_j for each neuron j. The weight corresponds to the position of the node in the high dimensional input space. The location of the neuron would be its position in the 2D latent space. SOM employs competitive learning such that during each iteration, only a single neuron wins a gradient from the input. SOM also employs a neighbourhood function such that after a neuron wins, a portion of the gradient is passed to its neighbouring neurons in the latent space. When fully trained, the competitive learning and the neighbourhood functions cause the SOM map to behave like a blanket being spread on a higher dimensional surface.

At the beginning of the SOM algorithm, the grid is initialized with random weights. Each iteration, a single sample is selected from the input and the node with the weight that has the smallest Euclidean distance is selected as the winning node:

$$c = argmin_j(||x[t] - w_j[t]||^2) \tag{1}$$

where c is the index of the winning node, $x[t]$ is the input at iteration t, and $w_j[t]$ is the weight of neuron j. The winning node and its neighbours receive a gradient that is inversely proportional to its distance in the latent space to the winner:

$$w_j[t+1] = w_j[t] + \alpha[t](\eta_{cj}[t](x[t] - w_j[t])) \tag{2}$$

where $\eta_{cj}[t]$ is the neighborhood function and $\alpha[t]$ is the learning rate. The neighborhood function is a monotonic decreasing function centered on the winning node. In this work we use a Gaussian window of a fixed size and shrinking σ as our neighbourhood function. Overall, this means that the winning node receives a large pull towards the input, while the neighbouring nodes receive a pull of diminishing strength as the nodes get further from the winner. By having the winner pass a portion of the gradient to its neighbours, the neighbours all start to share similar semantics after a few iterations.

While SOM is a type of neural network, its structure is not meant for deep learning. Previous works such as [12] and [23] have already combined SOM with deep learning models as explored in Section II-C, but in both of the works SOM was trained on the output of a single layer. In this work however, we wish to enforce a SOM-like locality on each layer of a CNN. That way, the filters of each layer can be arranged along the channel dimension and have its topology preserved. Since the filters of a CNN already receive gradients from back propagation, we decided to use those gradients rather than acquiring them from the input like in traditional SOM. Additionally, while we initially tried to incorporate competitive learning to our method, we found during our initial test that it significantly reduced the rate at which the network was learning and so it was taken out. In a future work, competitive learning could be added back in. Since the only part of SOM we are keeping is the neighbourhood function, we call our method LGNN to avoid confusion.

Before the weight update function from SOM can be used, it needs to be modified to fit the gradients from deep learning. In CNNs, the gradients now come from back propagation and are accumulated over batches and so we denote $g[k]$ as the gradient of the k^{th} filter which has been accumulated over a batch of images. We also note that since we have a batch of accumulated gradients instead of the gradient of a singular winner of an iteration, the neighbourhood function needs to shift and be applied to each of the gradients. From equation 2, we remove the time arguments and add a summation over k accumulated gradients in a batch. The new weight update function for each batch is:

$$\begin{aligned} w_{new}[j] &= w_{old}[j] + \alpha \sum_{k} (\eta[j-k]g[k]) \\ &= w_{old}[j] + \alpha(\eta * g)[j] \end{aligned} \tag{3}$$

As shown in equation 3, the term on the right is equivalent to performing a convolution between the accumulated gradients and the neighbourhood function. In other words, to achieve locality between filters, we simply reshape the channel dimension of the gradient matrix and then low pass filter it. The pseudocode for LGNN is as shown in Algorithm 1. The product of the SOM dimensions $[m, n]$ need to be equal the number of filters in the layer. Since each individual layer can have different number of output channels, we store the SOM dimensions as a lookup table.

Algorithm 1: Locality Guided Neural Networks

```
$c_{out}$ = output channels, $c_{in}$ = input channels, $s$ = filter size, $[m, n]$ = SOM dimensions
begin
    Initialize network
    Initialize LPF with fixed weights
    for each iteration do
        Zero gradients
        Input batch into network and calculate gradients from back propagation
        for each layer do
            Reshape gradient tensor:
              $[c_{out}, c_{in}, s, s] \rightarrow [(c_{in} \times s \times s), 1, m, n]$
            Apply LPF
            Reshape tensor back:
              $[(c_{in} \times s \times s), 1, m, n] \rightarrow [c_{out}, c_{in}, s, s]$
        Apply gradients
```

For our neighbourhood function, different LPF sizes were tested. It was found that when the LGNN switched from a 3×3 LPF to a 5×5 LPF, the validation accuracy dropped by more than 2%. We suspect that this is due to the fact that LGNN gets its gradients from back propagation rather than having the neurons matching the input distribution. While for SOM there is a singular input distribution, there are several local minimums that a network can settle on. If the pulling effect from the neighbourhood function is too strong, it may force the network to overfit to the training set and not generalize as well to new data.

There are several benefits to LGNN. Firstly, this method allows information to propagate locally within a layer. Normally, information in neural networks only propagates between layers, which can allow neurons within the same layer to carry redundant or conflicting information. The second benefit to LGNN is that applying it to a network only modifies the accumulated gradients during back propagation. No modifications are done to the network configuration nor the forward pass, and there is no required post-processing. Since back propagation is the only thing changed, an LGNN version of a network will have identical inference time to the baseline model. Also, since the accumulated gradients are the only tensors affected by LGNN, the added computational time of LGNN does not scale with image or batch size since the gradients are already accumulated for an entire batch.

IV. Experiments and Results

A. LGNN Applied to VGG and WRN

In our first experiment, we compare the performance of LGNN against their baseline models. For these experiments, CIFAR-100 is used as the dataset. We use two different baseline models: VGG [24] and Wide ResNet (WRN) [25]. For VGG, we use VGG-11 and VGG-19. For these models, we replace the last 3 FC layers with a single FC layer as [19] demonstrated that multiple FC layers adds too many network parameters without much benefit. For WRN, we use WRN-16-8 and WRN-28-10. All models use batch normalization [26] and the WRN models use the dropout version with 0.3 dropout. The optimizer is SGD with a momentum of 0.9 and a weight decay of 0.0005. VGG-11 ran for 100 epochs with an initial LR of 0.1 and decreased by a factor of 0.2 at epochs (40, 70, 90). The other 3 networks ran for 200 epochs with initial learning rate of 0.1 and decreased by a factor of 0.2 at epochs (60, 120, 160) just like in [25]. The reason why VGG-11 has different hyper parameters is because the model was small enough to finish training at 100 epochs and we wished to keep training time short for this model in order to test the different versions. For the neighbourhood function, a 3×3 Gaussian LPF was used with a σ of 0.5. Replication padding was used whenever the LPF was applied. In most of our experiments, two different version of the neighbourhood function were tested; in the first version the LPF stayed the same the whole time and in the second version, σ was changed at the rate of:

$$\sigma = 0.5 \times \left(1 - \frac{current\ epoch}{total\ epochs}\right) \tag{4}$$

In the results, we call the trials either constant or decreasing to refer to σ. Since LGNN does not change the network configuration or number of parameters, we were able to save 5 sets of random initializations and then ran each network configuration on each of those saves. When looking at the final results, we use the median instead of the mean as it is more robust against outliers when the number of trials is low.

VGG size	Regular	LGNN-Constant	LGNN-Decreasing
VGG-11	69.96	70.16	70.17
VGG-19	72.01	72.39	72.46

TABLE I: Accuracies (%) for VGG

The results for VGG-11 and VGG-19 are shown in table I. For the VGG networks, the median performances for all 3 networks are not dramatically different, with both versions of LGNN pulling slightly ahead of the baseline version. Also, having a decreasing or constant sigma did not significantly affect the median performance. Between the two sizes, VGG-19 had a larger performance increase. While the performance difference is minimal, we have demonstrated that training a network to have a local topology did not adversely affect the performance of the network.

Wide ResNets have a more complex structure than VGG with various convolution filter sizes and residual branches. Therefore, we test LGNN on various combinations of layers. In ResNets, the shortcut path is supposed to grant an easier path for gradients to flow to lower layers and so there is a possibility that applying LGNN to the 1×1 convolutions in them may decrease performance. Also, the very first convolution in WRN contains far fewer filters than the other layers and applying locality to such a small number of filters may harm the network's ability to learn distinct features. Therefore, the first configuration we use only applied LGNN to the main branch of the resblocks without changing the convolutions in the shortcuts and first layer. The second configuration applies LGNN to the main branch and the shortcuts in the resblocks but still ignores the first layer. The final configuration applies LGNN to all layers.

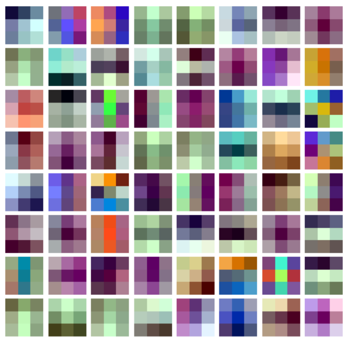
WRN size	Regular	Main Branch		ResBlocks		All	
		Con	Dec	Con	Dec	Con	Dec
16-8	78.87	78.90	78.94	79.08	79.10	79.07	79.12
28-10	80.62	-	-	80.88	81.08	80.99	80.81

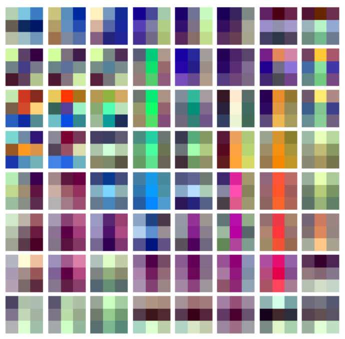
TABLE II: Accuracies (%) for Wide ResNet

The results on WRN are shown in table II. In the WRN-16-8 experiment, All LGNN configurations performed better than the baseline, with the *Main Branch* configuration performing slightly worse than the other two. The other LGNN configurations have relatively similar performances with the highest performance having a 0.25% increase compared to the baseline. The best median performance came from having a decreasing sigma where the difference between applying LGNN to only the resblocks or all layers was fairly minimal. In the WRN-28-10 experiment, since applying LGNN to only the main branch of the resblocks did not perform as well as the other two structures in WRN-16-8, those experiments were left out. The best median performance came from applying LGNN to only the resblocks with a decreasing sigma which had a performance increase of 0.46% increase.

Overall, while the performance increase was minimal, it scaled positively with network size. Our theory as to why LGNN marginally improves performance is due to regularization. While we use weight decay already, our other experiments in section IV-B suggests that the competitive learning from LGNN may have a regularizing effect on the weights. Another reason why LGNN could possibly have an effect on performance is that competitive learning could allow individual neurons to escape local minimums. In these situations, if a neuron is stuck in a local minimum, as long as one of its neighbours receives a large gradient, the neuron in question will receive a part of it and eventually escape the local minimum. The main focus of this work was to make CNNs more understandable while having minimal impact on the learning which was achieved by not having any of the median accuracies drop using our method.

B. Analysis of LGNN on the First Layer of a Network


(a) VGG-19 Baseline


(b) VGG-19 LGNN-Constant

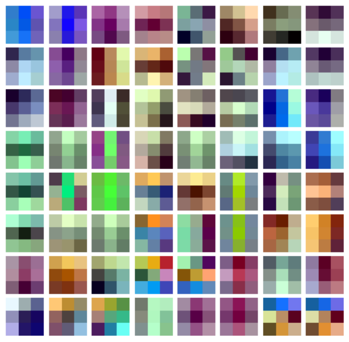

(c) VGG-19 LGNN-Decreasing

Fig. 1: Comparing the first layer of VGG-19 between the baseline and LGNN.

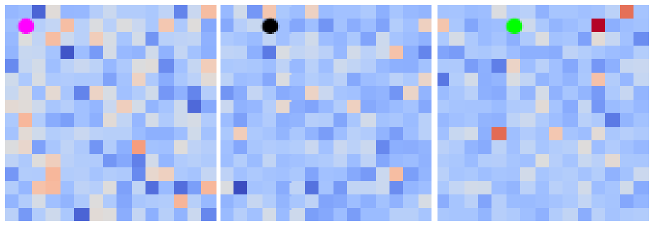
Magnitudes	Regular	LGNN-con	LGNN-dec
Min	$2.1775e-06$	0.0111	0.0007
Max	2.2977	1.6394	1.9456
Std Dev of Log	3.6854	1.0199	2.0902

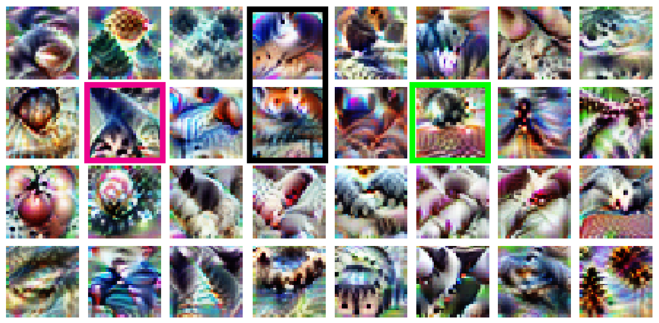
TABLE III: Magnitude statistics of filters in Figure 1

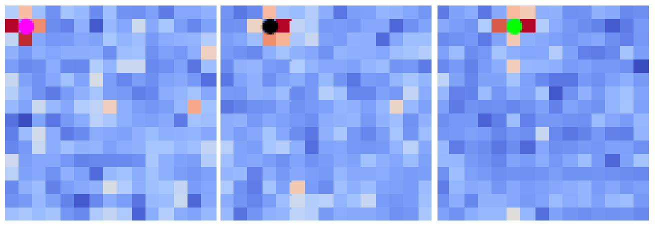
To further understand how LGNN affects a network, we examine the filters for the first layer of VGG-19 for one set of trials. The first layer for VGG-19 is used because it is the only layer with 3 input channels which makes it easier to visualize and VGG has more filters in the first layer. For the other layers, visualization techniques in [16] are required. The filter weights were normalized between [0 1] individually and then organized to the SOM dimensions. The first layer of VGG-19 contains 64 filters and the SOM dimensions were [8,8].

As shown in figure 1, LGNN has a noticeable effect on the learned filters. Compared to the baseline filters, the LGNN filters have an organized structure where the filters with similar appearances are grouped together locally. Looking at table III, the LGNN filters also have a smaller range of magnitudes. In the table, the smallest magnitude for the baseline is in the range of 10^{-6} while the LGNN with constant neighbourhood and decreasing neighbourhood have a minimum magnitude of 10^{-2} and 10^{-4}. This smaller range of magnitudes points at our method having a regularizing effect that reduces overfitting. It is important to note that since batch normalization and weight decay were being used in training there already was some regularization, but LGNN seems to still positively affect the training although minimally.

C. Analysis of LGNN on Hidden Layers of a Network


(a) Baseline Correlations


(b) Baseline Filter Visualizations


(c) LGNN-Constant Correlations

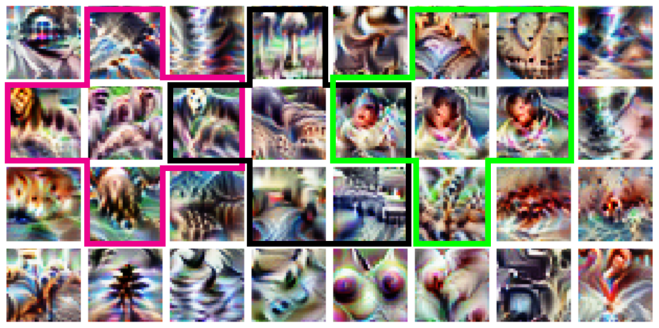

(d) LGNN-Constant Filter Visualizations

Fig. 2: Correlations between filters within the last layer of VGG-11.

While the first layer of a network is generally the only layer where weights can be directly mapped to color, the higher level layers are the ones that contain more interesting semantic filters. To visualize these filters, we use the techniques listed by [16] in which their code is available through Google's Lucid toolbox. The toolkit numerically generates images that maximally activates the filters. Our networks which were trained in Pytorch were transferred to Tensorflow using the Open Neural Network Exchange (ONNX) toolbox. Using the Lucid toolbox, the image size was set to 32 $\times$ 32, *fft* and *decorrelate* were set to *True*, and all other parameters were kept at their default values.

The first analysis we perform on the higher levels is to observe if the topology holds and if these correlations correspond to clusters of filters with similar semantic information. To see if the correlations hold, we first calculate a Gram matrix for the filter weights of the last layer of VGG-11. Each row of the Gram matrix represents the correlation between the filter corresponding to that row and all the other filters. By extracting said row and reshaping it to the SOM dimensions, we can then represent the correlations as a heat map. We set the main diagonal of the Gram matrix to 0 before extracting each row so the heat maps do not saturate. (a) and (c) of Figure 2 shows the heat maps of filters [1,1], [1,3], and [1,5]. We only show the top half of the heat maps to save space. Filter visualizations from Google's Lucid are also included in (b) and (d) to give readers a better understanding. Any correlated (red) neighbours in the heat maps have been outlined in the filter visualizations in their respective colors.

As shown in Figure 2, the LGNN-Constant version shows more correlations in a local area than the baseline version. In the heat maps for the baseline (a), most of the neighbours to each of the filters of interest are not highly correlated. Only the middle heat map has a single instance of a correlated neighbour. As for the heat maps for the LGNN version (c), all the direct neighbours of the indicated filters are highly correlated as shown by the red '+' in the heat map. The heat map in the middle and right even has one of the diagonals slightly correlated. The reason its not a red square is due to the choice of our LPF weights that we used in LGNN. Looking at (c) and (d), the filters in the outlined areas do indeed share some visual similarities, especially the filters outlined in green for (d) where the filters to the left and right are remarkably similar.

While the previous example demonstrates the correlation between neighbouring filters, the correlations may not necessarily be strong enough to assume that the filters will activate in clusters during inference. In Section II-B, we discussed attribution methods as a means of seeing how filters react to individual images. Unfortunately, getting an idea on how clusters of filters would react on average to various examples would be difficult for attribution methods. Instead, we look at the average activations for a full layer for a batch of images in one class. Since we have already shown in Figure 2 that the baseline network does not hold any correlation between neighbouring filters, we only show the activations for LGNN-

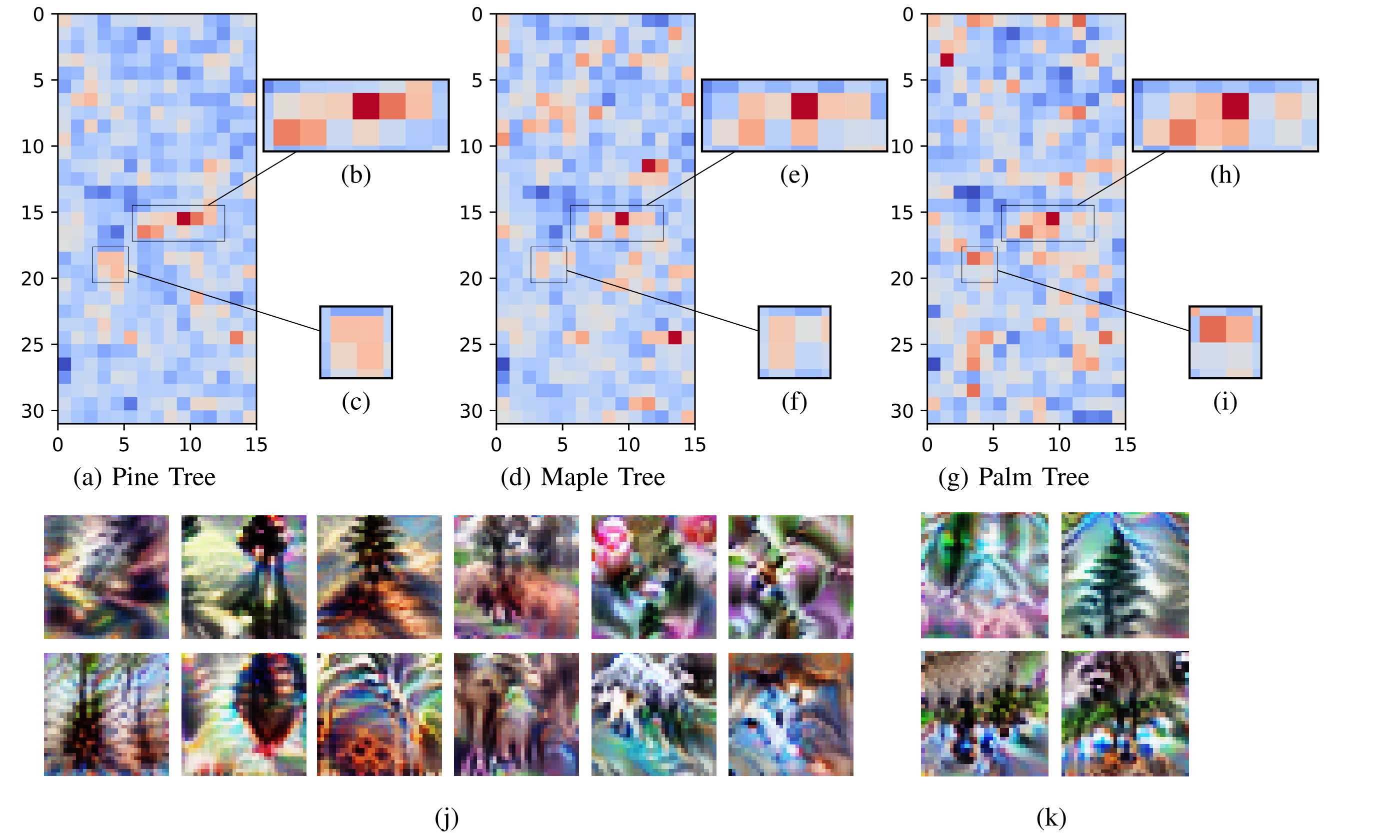


Fig. 3: Heat maps for average activations of the last convolution layer for 3 tree classes in CIFAR-100: Pine (a), Maple (d), and Palm (g) . For each class, activations were averaged over all 100 test images and then reshaped to SOM dimensions (32,16). Filter clusters shown by (b,e,h) and (c,f,i) seem to activate together for tree images. (j) and (k) show visualizations from Google's Lucid toolbox for (b,e,h) and (c,f,i) respectively

Constant. We take the 100 test images for each class for 3 classes in CIFAR-100 and feed them into the network. We then take the activations for the last convolution layer and average over the spatial and batch dimensions. The average activations are then mapped to the SOM dimensions and represented using a heat map.

In Figure 3, we first visualized all of the filters in the last layer, and found a cluster of filters that looked like trees shown in (k). Passing the classes (a) Pine Tree, (d) Maple Tree, and (g) Palm Tree we found that there were a few small positive activations in this cluster, but more importantly the region in (j) showed various filters in a larger cluster that also activated between various tree classes. Within the cluster in (j), one particular filter is the highest activation between the various trees while the other filters around it activate in differing proportions depending on the type of tree. It is important to note that the images fed into the network were the test images and not the training images. The test accuracies for these classes were 70% for the pine tree, 67% for the maple tree, and 90% for the palm tree. While observing the differences between activation maps for the incorrect and correct outputs would be interesting, it is outside the scope of this paper. One important detail is that the clusters of similar looking images correspond to correlated activations on average, and also the fact that despite having different output labels, 3 different tree classes had various positive activations for a specific cluster within the LGNN filters.

V. Conclusion

Our proposed method for XAI, LGNN is capable of gathering clusters of similar filters during training for a neural network. These clusters can often reveal to a user which filters respond to similar semantic concepts. Using the concept of competitive learning and neighbourhood functions which was inspired by SOM, our back propagation allows neurons within the same layer to share information with each other. One great benefit is that our system only modifies the back propagation and leaves the model structure and forward inference in tact. This makes our method fairly attractive if users wish to use our method for transfer learning where the trained network is being used only as a feature detector and a separate classifier is being trained that could possibly be hand crafted to take advantage of the organized filter structure.

Our experiments show reasonable success in making neural networks more explainable. In our first analysis we show that for different network types, the clustering from LGNN does not impede the learning capacity of the network and in fact offers a small accuracy increase on average. We believe this small accuracy improvement is due to the neighbourhood functions being a good regularizer. The reason why the accuracy improvement is fairly minimal could be due to the fact that the baseline networks already contained batch normalization and weight decay as regularizers already. In our second analysis, we observe the effect on LGNN within the first layer of a network. It was fairly evident that LGNN was capable of enforcing a topology onto the filters where clusters of neighbouring filters shared similar properties. We also compare the range of the magnitudes of the filters and see that LGNN does have a regularizing effect on the filters. In our last analysis, we examined the effect of LGNN on the last layer. The correlations between neighbouring filters still held up for LGNN in comparison to the baseline. We also managed to demonstrate that these correlations can lead to clusters that activate for similar semantic concepts, like how certain clusters activated for different types of trees even though the varieties of trees had different output labels.

Several future improvements to the system can be made. First of all, although our system takes concepts from SOM, we were not able to properly incorporate 'winner takes all' without it adversely affecting the learning. If winner takes all is reincorporated into the learning, it could possibly force similar filters to a singular cluster rather than several smaller clusters. Additionally, we can investigate tuning the hyper parameters for the neighbourhood function for each layer rather than sharing them between all layers.

References

[1] W. Rawat and Z. Wang, "Deep Convolutional Neural Networks for Image Classification: A Comprehensive Review," *Neural Computation*, vol. 29, no. 9, pp. 2352–2449, Sep. 2017.

[2] T. Young, D. Hazarika, S. Poria, and E. Cambria, "Recent Trends in Deep Learning Based Natural Language Processing [Review Article]," *IEEE Computational Intelligence Magazine*, vol. 13, no. 3, pp. 55–75, Aug. 2018. [Online]. Available: https://ieeexplore.ieee.org/document/8416973/

[3] "General Data Protection Regulation (GDPR) – Official Legal Text." [Online]. Available: https://gdpr-info.eu/

[4] X. Wang and A. Gupta, "Videos as Space-Time Region Graphs," in *European Conference on Computer Vision 2018*, ser. Lecture Notes in Computer Science, V. Ferrari, M. Hebert, C. Sminchisescu, and Y. Weiss, Eds. Cham: Springer International Publishing, 2018, pp. 413–431.

[5] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and F. F. Li, "ImageNet: a Large-Scale Hierarchical Image Database," in *IEEE Conference on Computer Vision and Pattern Recognition*, Jun. 2009, pp. 248–255.

[6] J. Carreira and A. Zisserman, "Quo Vadis, Action Recognition? A New Model and the Kinetics Dataset," in *proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, May 2017, pp. 6299–6308, arXiv: 1705.07750. [Online]. Available: http://arxiv.org/abs/1705.07750

[7] C. Olah, A. Satyanarayan, I. Johnson, S. Carter, L. Schubert, K. Ye, and A. Mordvintsev, "The Building Blocks of Interpretability," *Distill*, vol. 3, no. 3, p. e10, Mar. 2018. [Online]. Available: https://distill.pub/2018/building-blocks

[8] B. Kim, M. Wattenberg, J. Gilmer, C. Cai, J. Wexler, F. Viegas, and R. Sayres, "Interpretability Beyond Feature Attribution: Quantitative Testing with Concept Activation Vectors (TCAV)," in *International Conference on Machine Learning*, Jun. 2018, arXiv: 1711.11279. [Online]. Available: http://arxiv.org/abs/1711.11279

[9] D. Bau, B. Zhou, A. Khosla, A. Oliva, and A. Torralba, "Network Dissection: Quantifying Interpretability of Deep Visual Representations," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Honolulu, HI: IEEE, Jul. 2017, pp. 3319–3327. [Online]. Available: http://ieeexplore.ieee.org/document/8099837/

[10] M. T. Ribeiro, S. Singh, and C. Guestrin, ""Why Should I Trust You?": Explaining the Predictions of Any Classifier," in *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, Aug. 2016, arXiv: 1602.04938. [Online]. Available: http://arxiv.org/abs/1602.04938

[11] D. P. Kingma and M. Welling, "Stochastic Gradient VB and the Variational Auto-Encoder," in *International Conference on Learning Representations*, 2014, p. 14.

[12] V. Fortuin, M. Hüser, F. Locatello, H. Strathmann, and G. Rätsch, "SOM-VAE: Interpretable Discrete Representation Learning on Time Series," in *7th International Conference on Learning Representations (ICLR)*, May 2019, p. 18.

[13] P. Esser, E. Sutter, and B. Ommer, "A Variational U-Net for Conditional Appearance and Shape Generation," in *Conference on Computer Vision and Pattern Recognition*, Apr. 2018, arXiv: 1804.04694. [Online]. Available: http://arxiv.org/abs/1804.04694

[14] T. N. Kipf and M. Welling, "Semi-Supervised Classification With Graph convolutional Networks," in *International Conference on Learning Representations*, 2017, p. 14.

[15] K. Simonyan, A. Vedaldi, and A. Zisserman, "Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps," in *International Conference on Learning Representations*, Apr. 2014, arXiv: 1312.6034. [Online]. Available: http://arxiv.org/abs/1312.6034

[16] C. Olah, A. Mordvintsev, and L. Schubert, "Feature Visualization," *Distill*, vol. 2, no. 11, p. e7, Nov. 2017. [Online]. Available: https://distill.pub/2017/feature-visualization

[17] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra, "Grad-CAM: Visual Explanations from Deep Networks via Gradient-based Localization," *International Journal of Computer Vision*, Oct. 2019, arXiv: 1610.02391. [Online]. Available: http://arxiv.org/abs/1610.02391

[18] M. D. Zeiler and R. Fergus, "Visualizing and Understanding Convolutional Networks," in *European Conference on Computer Vision*, Nov. 2013, arXiv: 1311.2901. [Online]. Available: http://arxiv.org/abs/1311.2901

[19] J. T. Springenberg, A. Dosovitskiy, T. Brox, and M. Riedmiller, "Striving for Simplicity: The All Convolutional Net," in *International Conference on Learning Representations*, Apr. 2015, arXiv: 1412.6806. [Online]. Available: http://arxiv.org/abs/1412.6806

[20] Q. Zhang, Y. N. Wu, and S.-C. Zhu, "Interpretable Convolutional Neural Networks," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. Salt Lake City, UT: IEEE, Jun. 2018, pp. 8827–8836. [Online]. Available: https://ieeexplore.ieee.org/document/8579018/

[21] T. Kohonen, "Essentials of the self-organizing map," *Neural Networks*, vol. 37, pp. 52–65, Jan. 2013. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0893608012002596

[22] M. Kyan, G. Sun, H. Li, L. Zhong, P. Muneesawang, N. Dong, B. Elder, and L. Guan, "An Approach to Ballet Dance Training through MS Kinect and Visualization in a CAVE Virtual Reality Environment," *ACM Transactions on Intelligent Systems and Technology*, vol. 6, no. 2, pp. 1–37, Mar. 2015. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2753829.2735951

[23] M. Wang, W. Zhou, Q. Tian, J. Pu, and H. Li, "Deep Supervised Quantization by Self-Organizing Map," in *Proceedings of the 2017 ACM on Multimedia Conference - MM '17*. Mountain View, California, USA: ACM Press, 2017, pp. 1707–1715. [Online]. Available: http://dl.acm.org/citation.cfm?doid=3123266.3123415

[24] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," in *International Conference on Learning Representations 2015*, San Diego, CA, Sep. 2014, arXiv: 1409.1556. [Online]. Available: http://arxiv.org/abs/1409.1556

[25] S. Zagoruyko and N. Komodakis, "Wide Residual Networks," in *Procedings of the British Machine Vision Conference 2016*. York, UK: British Machine Vision Association, 2016, pp. 87.1–87.12. [Online]. Available: http://www.bmva.org/bmvc/2016/papers/paper087/index.html

[26] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *International Conference on Machine Learning*, 2015, pp. 448–456.