NEURON RANKING – AN INFORMED WAY TO COMPRESS CONVOLUTIONAL NEURAL NETWORKS

Anonymous authors
Paper under double-blind review

ABSTRACT

Convolutional neural networks (CNNs) in recent years have made a dramatic impact in science, technology and industry, yet the theoretical mechanism of CNN architecture design remains surprisingly vague. The CNN neurons, including its distinctive element, convolutional filters, are known to be learnable features, yet their individual role in producing the output is rather unclear. The thesis of this work is that not all neurons are equally important and some of them contain more useful information to perform a given task. Hence, we propose to quantify and rank neuron importance, and directly incorporate neuron importance in the objective function under two formulations: (1) a game theoretical approach based on *Shapley value* which computes the marginal contribution of each filter; and (2) a probabilistic approach based on what-we-call, the *importance switch* using variational inference. Using these two methods we confirm the general theory that some of the neurons are inherently more important than the others. Various experiments illustrate that learned ranks can be readily useable for structured network compression and interpretability of learned features.

1 INTRODUCTION

Neural networks have achieved state-of-the art results in various cognition tasks, including image and speech recognition, machine translation, reinforcement learning (Fergus et al., 2003; Mnih et al., 2013; Gu et al., 2018). Many of these applications involved CNNs which excel in particular in the vision tasks due to its ability to capture visual by means of convolution filters. Although the effectiveness of convolutional networks is unquestionable, the details of the architecture design and what particularly makes neural network work in detail remain highly uncertain. The experimental results roughly confirm that the accuracy of the network and representational capacity is correlated with the depth of the network (Simonyan & Zisserman, 2014; He et al., 2016; Montufar et al., 2014). Interestingly, the deeper architecture also become wider, although the link between width and network expressivity is questionable (Poole et al., 2016) and the choice of the number of neurons is rather discretionary. As a result the discussion about the network architecture often revolves around the numbers of filters and layers and their relative positioning, putting aside the conversation about the quality of the information that it contains.

The increasing size of the network architectures have faced scrutiny that made claims that the networks are overparametrized raising two main concerns: heavy computational load and potential overfitting (Louizos et al., 2017). In response to the need to build networks that are smaller yet accurate, a stream of research attempted to remove redundant units, compress the networks and design lighter architectures (Iandola et al., 2016; Ullrich et al., 2017). A widespread approach to network reduction has been removing weights that are small or even close to zero (Han et al., 2015). This line of research implicitly discerns that nodes with larger weights are more significant for learning task than the small weights. As a result, broadly speaking, this approach divides features between those that are useful which are kept and those which are insignificant and therefore discarded, forming a sort of binary approach.

In this work, we would like to scrutinize the individual filters and form an explicit theory that states that the units in the network (both convolutional filters and nodes in fully connected layers) are not equally important when it comes to performing an inference task. The corollary of this thesis is that CNNs learn features in a discriminative way so that some of them carry more significance than

others, and the knowledge about the input is not uniformly distributed among the CNN features. This theory is in line of research that adding more filters does not make the network more expressive since learning relevant information to the network has already been addressed by other filters.

Given the proposed theory, we would like to make a step forward in gaining insight what the CNN learns and propose to extend the binary approach to form a quantifiable ranking of features. In other words, we attempt to estimate the importance of each feature compared to the others with particular focus on convolutional filters, which may be visualized. We introduce a theoretical framework to quantify how important each feature is through proposing a feature ranking method based on two different approaches. The first approach derives from the game theoretical concept of *Shapley value* (Shapley, 1953), which assesses the importance of an individual in a group of neurons based on its marginal contribution to the group. The second method takes a probabilistic approach and introduces additional learnable parameters, which we call *importance switches*, that take real values and are trained by means of variational inference to give more weight to the important features. The extensive experimental results using these approaches indicate that some features are inherently more significant than others.

The theoretical underpinnings of the feature rankings have further direct practical implications we explore. Firstly, the knowledge of the ranking allows to know which features directly impact the score of our method and consequently a more informed way of building an effective model. Thus, we are able to build a network around the the relevant features and discard the less relevant ones, effectively compressing the network achieving state-of-the-art results. Secondly and perhaps more significantly, the feature ranking of convolutional features provides more interpretable information about the network and places meaning on particular features in the context of a given task, thus casting light on the black box models. To achieve human interpretability, we visualize the most significant features which significantly show the significance of repeated and complementary features.

RELATED WORK

In early years of CNN development, the networks were limited to a few layers (LeCun et al., 1998). Recently, the architectures have become deeper and wider (Krizhevsky et al., 2012; Szegedy et al., 2015). The emergence of GPU implementability and regularization algorithms (Srivastava et al., 2014; Ioffe & Szegedy, 2015) has allowed to use large architectures which train and generalize well. Nevertheless, the trend towards building larger neural networks ironically opposed the research about the nature, interpretability and knowledge extraction from within the neural network models, which we are interested in this work. Therefore, we will compare our method to existing ones in terms of compression ability and interpretability, and then frame the idea in terms of neuron ranking.

Compression. The early work on compression largely focused on non-Bayesian approaches, e.g., (Hassibi & Stork, 1993) and mostly centered around non-structured pruning methods, e.g., removing single weights from the architectures of CNNs (Han et al., 2015). Then, hardware-oriented structured pruning techniques made more practical speed-ups (Srinivas & Babu, 2015; Li et al., 2016; Wen et al., 2016; Lebedev & Lempitsky, 2016; Zhou et al., 2016). More recently Bayesian methods using the network weights' uncertainties have achieved impressive compression rates, e.g., using sparsity inducing prior on scale parameter (Molchanov et al., 2017), using Gaussian mixture priors (Ullrich et al., 2017), and using the grouping of weights through a group Horseshoe prior (Louizos et al., 2017), among many. However, none of these methods prune neurons based on the direct *optimization for extracting the importance of each neuron*.

Interpretability. Broadly there are three lines of work done for intepretability of CNNs. The first line of work, the early and used-to-be very popular work, focused on visualization of neurons in CNNs to understand how information propagates within the network (Zeiler & Fergus, 2013; Simonyan et al., 2014; Yosinski et al., 2015). Another line of work focused on probing trained CNNs to obtain local & pixel level explanations either layer-wise or class-wise using gradient information of querying points (Selvaraju et al., 2016; Bach et al., 2015; Montavon et al., 2015). Last line of work focused on mapping semantic concepts to latent representations of CNNs (Bau et al., 2017). Other somewhat related work for interpretability using Shapley value also exist (but not in the context of CNNs) (Lundberg & Lee, 2017). Compared to existing methods, our method provides a *global view of a trained model* in terms of the importance of learned features.

In what follows, we introduce the two methods to perform neuron ranking.

2 GAME THEORETICAL NEURON RANKING

The first approach derives from the game theoretical concept of *Shapley value* (Shapley, 1953). The concept allows to compute the importance score (payoff) of an individual based on the payoffs given to collections of individuals. We subsequently adapt the concept to rank the neurons in terms of their predictive utility. Assuming that an important feature allows for task generalization, the feature importance further translates into finding features that carry most information and usefulness in the prediction task that lead to achieving higher accuracy.

2.1 COALITIONAL GAME THEORY

A coalitional game is a game where utility is given to a group of players (in our case, nodes or neurons) instead of each agent individually. Let N be the number of agents, which in this work are CNN features (also referred as neurons or nodes). To be specific, let N_l to be the number of neurons in a layer l (in unambiguous cases, for clarity we omit the subscript). For every group of players, a coalitional game specifies the payoff the members receive as a group or a coalition. We define a *coalition* of the neurons N of a layer L as a subset of neurons, $C \subseteq N$. To assess quantitatively the performance of a group of agents, each coalition is assigned to a real number, which is interpreted as a payoff that a coalition receives from being together. Mathematically, the value of a coalition is given by a *characteristic function*, which assigns a real number to a set of nodes. Formally, a characteristic function ν: $2^N \rightarrow \mathbb{R}$ maps each coalition (subset) $C \subseteq N$ to a real number $\nu(C)$. Therefore, a coalitional game is defined by a tuple (N, ν), where N is a set of players and ν is a function that assigns payoffs to every coalition of N.

A critical component of a coalitional game is specifying the choice of characteristic function that is assigned to a given subset of features. In the case of CNN, the test metric is accuracy which assesses whether the (argmax of) the network output is the correct label averaged over the number of examples. As a result, we choose the accuracy on a validation set as the characteristic function, that is, $\nu(C) = \text{acc}(C)$ and $\nu(N) = \text{acc}(N)$. The question now remains how to assess the importance of a single feature given the information about the payoffs for each subset of nodes. To this end, we employ the concept of Shapley value about the normative payoff of the total reward/cost, that is a division scheme that allows to distribute the total payoff uniquely and in a fair way.

2.2 SHAPLEY VALUE

Shapley proposes to evaluate each player by the marginal contribution that the player makes to every coalition averaged over all the coalitions. The marginal contribution of an agent n is the difference between the value of a coalition C that contains n and the coalition $C \setminus n$. For example, when a coalition has no members, i.e. is empty, and the neuron n_1 joins the coalition, the value of its marginal contribution is equal to the value of the one-member coalition as the value of the empty coalition is equal to 0, $\nu(\{n_1\}) - \nu(\{\emptyset\}) = \nu(\{n_1\})$ where $\{n_1\} = C$. Subsequently, when another agent n_2 joins this coalition, its marginal contribution is equal to $\nu(\{n_1, n_2\}) - \nu(\{n_1\})$. The process continues until all the nodes join the coalition. The coalition of all the nodes is called the *grand coalition*.

The order of nodes, which builds subsequent coalitions to finally the grand coalition, can be represented as a permutation of nodes. For example, in the case of permutation $n_5 n_3 n_7 ... n_N ... n_2$, the neuron n_5 creates the first non-empty coalition on it own, and we measure the accuracy of the pretrained model which includes only one neuron, n_5, in the given layer of the original pre-trained model. Then two-element coalition $n_5 n_3$ is formed corresponding to the two-neuron layer, and so on. All the subsequent nodes join the coalition in the order given by the permutation. There are $N!$ permutations of N nodes, meaning that there are $N!$ different ways to form a coalition. To compute the Shapley value of the node n, we compare the accuracy of the architecture before and after adding the node n, that is the marginal contributions of n (which may be negative) for each of the $N!$ permutations and divide the sum by all the permutations. The Shapley value of n is then the averaged marginal contribution of n.

Formally, let π denote a permutation, $\pi(i)$ a place of the neuron n_i in the permutation π, and $C_\pi(i)$ the coalition formed by the predecessors of n_i such that $C_\pi(i) = \{n_j \in \pi : \pi(j) \text{ before } \pi(i)\}$. For example, in the permutation $n_5 n_3 n_7 ... n_N ... n_2$, $\pi(3) = n_7$ and $C_\pi(3) = \{n_5, n_3\}$. The Shapley

value (SV_i) of the node n_i is thus defined as follows:

$$SV(n_i) = \sum_{\pi \in \Pi(N)} \frac{1}{|N|!}(\nu(C_\pi(i) \cup \{n_i\}) - \nu(C_\pi(i))) \tag{1}$$

This formula can also be written in a form that considers sets instead of permutations:

$$SV(n_i) = \sum_{C \subseteq N \setminus \{v_i\}} \frac{|C|!(|N| - |C| - 1)!}{|N|!}(\nu(C \cup \{n_i\}) - \nu(C_\pi(i))) \tag{2}$$

PRACTICAL CONSIDERATIONS

First, Shapley value is a mathematically rigorous division scheme and, strictly speaking, it has been proposed as the only measure that satisfies four normative criteria regarding the fair payoff distribution. These criteria are (1) efficiency where the total gain is distributed among the agents, (2) symmetry; if i and j are agents such that $\nu(C \cup i) = \nu(C \cup j)$ for each coalition C of N, then $SV(i) = SV(j)$, (3) null player payoff such that an agent who contributes nothing to every coalition obtains zero individual payoff and (4) linearity; $\nu(C) = \nu_1(C) + \nu_2(C)$ for every coalition implies $SV_{\nu_1}(i) + SV_{\nu_2}(i) = SV_\nu(i)$. Nevertheless, the choice of a characteristic function which satisfies these criteria is not feasible in case of our application due to the fact that we do not have control over the output of the model. As a result, the characteristic function may not be monotone which violates the first criterion. However, the payoff produced by the Shapley value, although may not be unique, is a valid cost division which works well in practice.

Second, computing the characteristic function for every subset is combinatorial and takes exponential time complexity. Hence, for large networks, computing Shapley value is computationally infeasible. We propose the following solutions to approximate the optimal solution and obtain a sensible ranking metric based on Shapley value. The first solution entails computing the Shapley value for the subsets no larger than arbitrary k. As a result we only compute the synergies that are no larger than k. Intuitively, we assume that that the larger the coalition, the less information is to be obtained from computing the large subsets. The second solution is based on sampling and sampling provides an unbiased estimate of the optimal result. Thus, we first sample the characteristic function and then sample the permutations needed for the computations of the Shapley value.

What comes next describes our proposal to improve the speed of computation for identifying the neuron ranking in a continuous manner.

3 PROBABILISTIC NEURON RANKING

3.1 IMPORTANCE SWITCHES

To infer the neuron ranking in each layer, we propose to make a slight modification in the existing neural network architecture. We introduce a component, the *importance switch*, denoted by $\boldsymbol{s}_l$ for each layer l. Each importance switch is a probability vector of length D_l (the output dimension of the lth layer) and $\sum_j^{D_l} \boldsymbol{s}_{l,j} = 1$, where $\boldsymbol{s}_{l,j}$ is the jth element of the vector. With this addition, we rewrite the forward pass under a deep neural network model, where the function $f(\boldsymbol{W}_l, \boldsymbol{x}_i)$ can be the convolution operation for CNNs or simple matrix multiplication for MLPs between the weights $\boldsymbol{W}_l$ and the unit $\boldsymbol{x}_i$,

$$\text{Pre-activation followed by a switch } \boldsymbol{s}_l\text{: } \boldsymbol{h}_{l,i} = \boldsymbol{s}_l \circ [f(\boldsymbol{W}_l, \boldsymbol{x}_i)]\,, \tag{3}$$

$$\text{Input to the next layer after going through a nonlinearity } \sigma\text{: } \boldsymbol{z}_{l,i} = \sigma(\boldsymbol{h}_{l,i}), \tag{4}$$

where $\circ$ is an element-wise product. Introducing a switch operation between layers in a neural network model was also presented in (Louizos et al., 2017), although in their case, the switch is a binary random variable (called a gate). The output probability under such networks with L hidden layers for solving classification problems can be written as

$$P(\boldsymbol{y}_i|\boldsymbol{x}_i, \{\boldsymbol{W}_l\}_{l=1}^{L+1}) = g\left(\boldsymbol{W}_{L+1}\boldsymbol{z}_{L,i}\right), \text{ where } \boldsymbol{z}_{L,i} = \sigma(\boldsymbol{s}_L \circ [f(\boldsymbol{W}_L\boldsymbol{z}_{L-1,i})]). \tag{5}$$

where g is the *softmax* operation.

3.2 VARIATIONAL LEARNING OF IMPORTANCE SWITCHES

A natural choice to model the distribution over the switch is the *Dirichlet* distribution, which defines a probability distribution over a probability vector. We model each switch as a vector of independent Dirichlet distributed random variables

$$p(\boldsymbol{s}_l) = \text{Dir}(\boldsymbol{s}_l|\boldsymbol{\alpha}_0). \tag{6}$$

When there is no prior knowledge, i.e., *a priori* we don't know which feature would be more important for prediction, so we treat them all equally important features by setting the same value to each parameter, i.e., $\boldsymbol{\alpha}_0 = \alpha_0 * \mathbf{1}_{D_l}$ where $\mathbf{1}_{D_l}$ is a vector of ones of length D_l. When we apply the same parameter to each dimension, this special case of Dirichlet distribution is called *symmetric* Dirichlet distribution. In this case, if we set $\alpha_0 < 1$, this puts the probability mass toward a few components, resulting in only a few components that are non-zero, i.e., inducing sparse probability vector. If we set $\alpha_0 > 1$, all components become similar to each other.

We model the posterior over $\boldsymbol{s}_l$ as the Dirichlet distribution as well but with *asymmetric* form to learn a different probability on different elements of the switch (or neurons), using a set of variational parameters (the parameters for the posterior). We denote the variational parameters by $\boldsymbol{\phi}_l$, where each element of the vector can choose any values above 0. Our posterior distribution over the switch is, hence, defined by

$$q_{\boldsymbol{\phi}_l}(\boldsymbol{s}_l) = \text{Dir}(\boldsymbol{s}_l|\boldsymbol{\phi}_l). \tag{7}$$

With this parametric form of prior and posterior, we optimize the variational parameters $\boldsymbol{\phi}_l$ over each layer's importance switch by maximizing the variational lower bound with freezing all the weights to the pre-trained values,

$$\log p(\mathcal{D}) \geq \mathcal{L}(\boldsymbol{\phi}_l) := \int q_{\boldsymbol{\phi}_l}(\boldsymbol{s}_l) \log p(\mathcal{D}|\boldsymbol{s}_l) d\boldsymbol{s}_l - \text{D}_{kl}[q(\boldsymbol{s}_l|\boldsymbol{\phi}_l)||p(\boldsymbol{s}_l|\boldsymbol{\alpha}_0)]. \tag{8}$$

We do this variational learning for each layer's importance switch sequentially from the input layer to the last layer before the output layer.

Computing the gradient of equation 8 with respect to $\boldsymbol{\phi}_l$ requires computing the gradients of the integral (the first term on RHS) and also the KL divergence term (the second term on RHS), as both depends on the value of $\boldsymbol{\phi}_l$. The KL divergence between two Dirichlet distributions can be wrttien in closed form. However, the first term is tricky. As described in (Figurnov et al., 2018), the usual reparameterization trick, i.e., replacing a probability distribution with an equivalent parameterization of it by using a deterministic and differentiable transformation of some fixed base distribution[1], does not work. For instance, in an attempt to find a reparameterization, one could adopt the representation of a k-dimensional Dirichlet random variable as a weighted sum of Gamma random variables, $\boldsymbol{s}_{l,j} = y_j/(\sum_{j'=1}^{K} y_{j'})$, where $y_j \sim \text{Gam}(\boldsymbol{\phi}_{l,j}, 1) = y_j^{(\boldsymbol{\phi}_{l,j}-1)} \exp(-y_j)/\Gamma(\boldsymbol{\phi}_{l,j})$, for $\boldsymbol{s}_l \sim \text{Dir}(\boldsymbol{s}_l|\boldsymbol{\phi}_l)$, where the shape parameter of Gamma is $\boldsymbol{\phi}_{l,j}$ and the scale parameter is 1. However, this does not allow us to detach the randomness from the parameters as the parameter still appears in the Gamma distribution, hence one needs to sample from the posterior every time the variational parameters are updated, which is costly and time-consuming. Existing methods suggest either explicitly or implicitly computing the gradients of the inverse CDF of the Gamma distribution during training to decrease the variance of the gradients (e.g., (Knowles, 2015) and (Figurnov et al., 2018) among many). The length of the importance switch we consider is mostly less than on the order of 100s, in which case the variance of gradients does not affect the speed of convergence as significantly as in other cases such as Latent Dirichlet Allocation (LDA). Hence, when training for the importance switch in each layer, we use the analytic mean of the Dirichlet random variable to make a point estimate of the integral $\int q_{\boldsymbol{\phi}_l}(\boldsymbol{s}_l) \log p(\mathcal{D}|\boldsymbol{s}_l) d\boldsymbol{s}_l \approx \log p(\mathcal{D}|\tilde{\boldsymbol{s}}_l)$, where $\tilde{\boldsymbol{s}}_{l,j} = \boldsymbol{\phi}_{l,j}/\sum_{j'=1}^{D_l} \boldsymbol{\phi}_{l,j'}$, which allows us to directly compute the gradient of the quantity without sampling from the posterior. As illustrated in section 4, this approximation performs well with relatively low dimensional switches.

[1]For instance, a Normal distribution for z with parameters of mean μ and variance σ^2 can be written equivalently as $z = \mu + \sigma\epsilon$ using a fixed base distribution $\epsilon \sim (0, 1)$.

Game-theoretic vs. probabilistic neuron ranking How are the game-theoretic and the probabilistic neuron ranking methods related? Consider a vector of random variables $\boldsymbol{r}$ that describes a certain ranking for a certain number of neurons. The predictive distribution on the test data $\mathcal{D}^*$, in this case, can be obtained by integrating out a plausible distribution over the neuron rankings, which we denote by $f(\boldsymbol{r})$,

$$p(\mathcal{D}^*|\mathcal{D}) = \int p(\mathcal{D}^*|\boldsymbol{r}, \mathcal{D}) f(\boldsymbol{r}) d\boldsymbol{r}, \tag{9}$$

$$\approx p(\mathcal{D}^*|\hat{\boldsymbol{r}}, \mathcal{D}), \tag{10}$$

where the second line is a reasonable approximation if the distribution is highly peaked at around the optimal ranking $\hat{\boldsymbol{r}}$, meaning that there is indeed such an optimal ranking with a high confidence. This predictive distribution specifies the likelihood of the test data given that optimal ranking. In the multi-class classification, this predictive distribution is the likelihood of true lables given a classifier's predictions. When we compute the Shapley value, we use an "approximate" version of this predictive likelihood, namely, we introduce a max operation for choosing a single label that has the maximum probability for that class, and then see if the label matches the true label, resulting in the frequency of correct labeling as an accuracy measure. Hence, both methods attempt to find the best ranking in order to "maximize the likelihood of data". While the Shapley optimization attempts to maximize the test data likelihood approximately in a combinatorial manner, the switch optimization attempts to maximize the training data likelihood with a regularization as in equation 8 in a continuous manner.

4 EXPERIMENTS

In this section we present experimental results based on the two proposed approaches for CNN features ranking, the Shapley value and the importance switch methods. The tests have been performed on LeNet-5 trained on MNIST and FashionMNIST, and VGG-16 trained on CIFAR-10.

To compute the rankings for both methods the same pretrained model is used. To compute the Shapley value of each neuron in the trained model, we remove the subsets of features (both weights and biases) and test the network on a validation set. As mentioned, the accuracy is the payoff for a given group of features. The computation of the complete set of payoffs is of combinatorial nature and therefore we compute the power set for layers up to 25 nodes. To account for this limitation and to illustrate better the proposed method, we choose to limit the number of nodes in the pretrained LeNet-5 architecture to 10-20-100-25. When using the trained VGG-16, we use the same number of filters in each layer as in the original architecture. For the layers with larger number of features, we use one of the two methods to compute marginal contributions. The first method uses equation 1 and only limits the number of coalitions we consider to compute SV. The second method uses equation 2 the accuracy change between two subsets which differ by a single node. Both node and the first combination were sampled uniformly at random.

When we learn the importance switches, we load the same train model which has been used to compute the Shapley value and then only add parameters for switches and trained them per layer with fixing all the other network parameters to the trained values. We run the training of the importance switches for 300 epochs, however, in practice, even a few iterations is sufficient to distinguish important nodes from the rest.

Method comparison: We start with comparing the learnt ranks of the two methods. As summarized in Table 1, the first observation is that for the model pretrained both on MNIST and FashionMNIST both methods have identified similar nodes to be the most important. The similarity is more significant for smaller layers where over 50% of top nodes (here we consider top-5 nodes for clarity and top-10 nodes for the large fc1 layer) and in three out of six cases the top two nodes are the same. Significantly for conv2 on MNIST the group of four nodes are the same, and as far as fc2 on FashionMNIST is concerned, the top five nodes chosen from the set of 25 nodes are the same (the probability to select this subset at random is $6 \cdot 10^{-5}$), showing that the methods agree when it comes to both convolutional and fully connected layers. For brevity, please look at the Appendix for the rankings of the less significant nodes but what is notable is that both methods also identified similar groups of unimportant nodes, particularly in fc2 where every node indexed higher than 9 (as compared to nodes indexed lower than 9) scored very low for both methods. When it comes to larger

layers, the methods however are more discrepant (yet still significantly the common nodes are found as seen in the case of fc1 layer). The differences may also come from the inexact computation of the Shapley value.

Layer	Alg	FashionMNIST	MNIST
conv1 (10)	SH	**0**,**7**,**6**,**5**,1	**1**,**8**,7,4,**6**
	IS	**0**,**7**,**5**,9,**6**	**8**,**1**,3,9,**6**
conv2 (20)	SH	**5**,10,0,**13**,9	**2**,**8**,**9**,**19**,4
	IS	**5**,8,**13**,14,15	**9**,**2**,**8**,**19**,6
fc1 (100)	SH	**60**, **13**, 43, 88, 94, 20, 70, 44, 32, 64	**56**, 86, 25, 64, 33, 17, **23**, **96**, 52, 81
	IS	94, 7, 50, 92, **13**, 25, **60**, 40, 75, 45	25, **96**, 58, **56**, 88, 52, **23**, 43, 30, 4
fc2 (25)	SH	**5**,**1**,**8**,**9**,**7**	**1**,**7**,2,3,0
	IS	**1**,**7**,**9**,**5**,**8**	**7**,**1**,4,6,9

Table 1: Rankings of filters for the Shapley value (SH) and the importance switches (IS) methods on a four-layer network, 10-20-100-25. For each layer the top five neurons are shown, the numbers in bold indicate the common top neurons across both the methods.

Interpretability: One of the main aims of this work has been to understand better the process of learning of convolutional neural networks. Building on the previous works which visualized CNN filters, we want to add an extra component and interpret that visual features by means of the filter rankings. In figure 1, we visualize feature maps produced by the first convolution layer of filters. Knowing the important filters allows to ponder over what features the network learns and deems useful. For instance, in the MNIST digits, the learnt filters identify local parts of the image (such as lower and upper parts of the digit '2' and opposite parts of the digit '0'). The interesting observation is that the most important features, on the one hand, complement each other (such as complementing parts of the digit '0' or the dog in CIFAR-10) but, on the other and, overlap to seemingly reinforce its importance. Finally, the important features appear smoother as compared to unimportant ones, which outline the object with no particular focus.

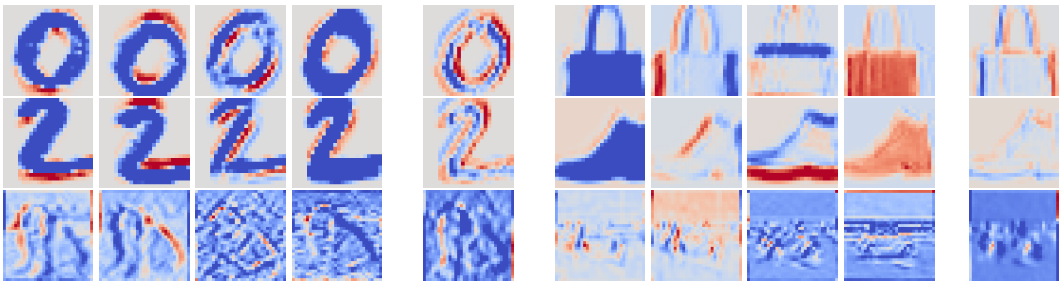

Figure 1: Visualization of four important feature maps (MNIST: 1,8,3,7, FashionMNIST: 0,7,5,6) and one unimportant one for two examples of digits for the same filters. The third row depicts the feature maps from CIFAR-10. Notice the complementary nature of important features on MNIST: (1,3), FashionMNIST (0,7) and CIFAR-10 (the first two) and reinforcing features: MNIST (1,8), FashionMNIST (7,5). The unimportant features are jagged or lack concrete focus.

Compression: The consequence of the feature ranking is that some of the nodes within each layer are less significant than others and, as argued in network compression literature, the network may do as well without them. The compression experiments procedure follows from the previous experiments. Given the rankings, we prune the neurons from the bottom of the ranking and then we retrain the network. We run the tests for both of the methods on several different architectures. In all the trainings we use SGD with decreasing learning rate from 0.1 to 0.001, momentum, 0.9, weight decay, 5e-4, and early stopping.

LeNetVGG presents the results for LeNet-5 as trained on MNIST, and VGG-16 as trained on CIFAR-10. For LeNet-5, the compressed architecture has 17K parameters which is less than all the other methods, and 137K FLOPs which is second to FDOO(100) (Tang et al., 2018), which however has over three times more parameters. The method fares relatively well also on VGG producing an

architecture which is smaller than others in the earlier layers but larger in later layers (the second proposed architecture has overall the least number of parameters at the cost of the performance, though). We hope to test a larger set of possible architectures in the future and devise a way to combine both rankings for a more optimal compression. Nevertheless, the results show that the neuron ranking method is adequate for condensing both small and large architectures.

Method	Architecture	Error	FLOPs	Params
NR (proposed)	**5-7-45-20**	**1.0%**	**130K**	**4K**
BC-GNJ	8-13-88-13	1.0%	284K	11K
BC-GHS	5-10-76-16	1.0%	155K	8K
FDOO(100K)	2-7-112-478	1.1%	111K	242K
FDOO(200K)	3-8-128-499	1.0%	153K	267K
GL	3-12-192-500	1.0%	205K	288K
GD	7-13-208-16	1.1%	253K	46K
SBP	3-18-284-283	0.9%	217K	163K
NR (proposed)	**34-34-68-68-75-106-101-92-102-92-92-67-67-62-62**	**8.6%**	**44M**	**1.3M**
NR (proposed)	**39-39-63-48-55-98-97-52-62-22-42-47-47-42-62**	**9.1%**	**34M**	**0.9M**
BC-GNJ	63-64-128-128-245-155-63-26-24-24-20-14-12-11-15	8.3%	142M	1.0M
BC-GHS	51-62-125-128-228-129-38-13-9-6-5-6-6-20	8.3%	122M	0.8M

Table 2: The structured pruning of LeNet-5 and VGG-16

The final experiment demonstrates how our method compares to magnitude pruning commonly done in compression literature. In figure 2, our method (blue trace) outperforms magnitude pruning methods (L1 and L2 norm over weights). No retraining is used in this case to show how the proposed method retains the relevant neurons that affect the predictive accuracy.

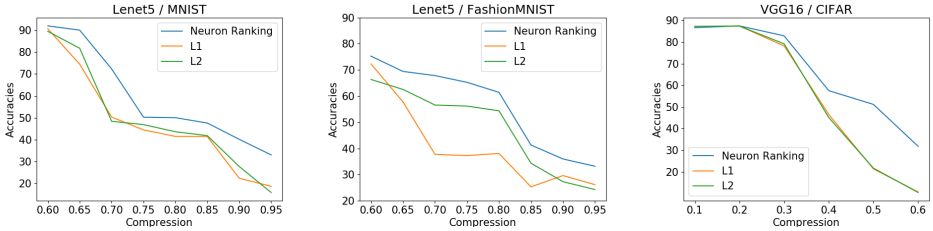

Figure 2: Comparison to magnitude pruning

We would like to emphasize that the magnitude approaches may be more appropriate for the unstructured pruning where single weights are removed based on its magnitude. However, in the the case of pruning entire channels, considering a norm of weights may be too simplistic as the interactions between the weights within a channel are rather complex. The proposed new paradigm treats the channels as whole units that directly contribute to the task generalization.

5 Conclusion

In summary, this work suggests a theory that the learnable CNN features contain inherent hierarchy where some of the features are more significant than others. This multidisciplinary work which builds on top of probability and game theoretical concepts proposes two methods to produce feature ranking and select most important features in the CNN network. The striking observation is that the different methods lead to similar results and allow to distinguish important nodes with larger confidence. The ranking methods allow to build an informed way to build a slim network architecture where the significant nodes remain and unimportant nodes are discarded. A future search for further methods which allow to quantify the neuron importance is the next step to develop the understanding of the feature importance in CNNs.

REFERENCES

Sebastian Bach, Alexander Binder, Grgoire Montavon, Frederick Klauschen, Klaus-Robert Mller, and Wojciech Samek. On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation. *PLOS ONE*, 10(7):1–46, 07 2015. doi: 10.1371/journal.pone.0130140. URL https://doi.org/10.1371/journal.pone.0130140.

David Bau, Bolei Zhou, Aditya Khosla, Aude Oliva, and Antonio Torralba. Network dissection: Quantifying interpretability of deep visual representations. *CoRR*, abs/1704.05796, 2017. URL http://arxiv.org/abs/1704.05796.

R. Fergus, P. Perona, and A. Zisserman. Object class recognition by unsupervised scale-invariant learning. In *2003 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2003. Proceedings.*, volume 2, pp. II–II, June 2003. doi: 10.1109/CVPR.2003.1211479.

Mikhail Figurnov, Shakir Mohamed, and Andriy Mnih. Implicit reparameterization gradients. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett (eds.), *Advances in Neural Information Processing Systems 31*, pp. 441–452. Curran Associates, Inc., 2018. URL http://papers.nips.cc/paper/7326-implicit-reparameterization-gradients.pdf.

Jiuxiang Gu, Zhenhua Wang, Jason Kuen, Lianyang Ma, Amir Shahroudy, Bing Shuai, Ting Liu, Xingxing Wang, Gang Wang, Jianfei Cai, and Tsuhan Chen. Recent advances in convolutional neural networks. *Pattern Recognition*, 77:354 – 377, 2018. ISSN 0031-3203. doi: https://doi.org/10.1016/j.patcog.2017.10.013. URL http://www.sciencedirect.com/science/article/pii/S0031320317304120.

Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015.

Babak Hassibi and David G Stork. Second order derivatives for network pruning: Optimal brain surgeon. In *Advances in neural information processing systems*, pp. 164–171, 1993.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Forrest N Iandola, Song Han, Matthew W Moskewicz, Khalid Ashraf, William J Dally, and Kurt Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and¡ 0.5 mb model size. *arXiv preprint arXiv:1602.07360*, 2016.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

David A. Knowles. Stochastic gradient variational Bayes for gamma approximating distributions. *arXiv e-prints*, art. arXiv:1509.01631, Sep 2015.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pp. 1097–1105, 2012.

Vadim Lebedev and Victor Lempitsky. Fast convnets using group-wise brain damage. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun 2016. doi: 10.1109/cvpr.2016.280. URL http://dx.doi.org/10.1109/CVPR.2016.280.

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

Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. *arXiv preprint arXiv:1608.08710*, 2016.

Christos Louizos, Karen Ullrich, and Max Welling. Bayesian compression for deep learning. In *Advances in Neural Information Processing Systems*, pp. 3288–3298, 2017.

Christos Louizos, Max Welling, and Diederik P. Kingma. Learning Sparse Neural Networks through L_0 Regularization. *arXiv e-prints*, art. arXiv:1712.01312, Dec 2017.

Scott M Lundberg and Su-In Lee. A unified approach to interpreting model predictions. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (eds.), *Advances in Neural Information Processing Systems 30*, pp. 4765–4774. Curran Associates, Inc., 2017. URL http://papers.nips.cc/paper/7062-a-unified-approach-to-interpreting-model-predictions.pdf.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013. URL https://arxiv.org/pdf/1312.5602.pdf.

Dmitry Molchanov, Arsenii Ashukha, and Dmitry Vetrov. Variational dropout sparsifies deep neural networks. *arXiv preprint arXiv:1701.05369*, 2017.

Grégoire Montavon, Sebastian Bach, Alexander Binder, Wojciech Samek, and Klaus-Robert Müller. Explaining nonlinear classification decisions with deep taylor decomposition. *CoRR*, abs/1512.02479, 2015. URL http://arxiv.org/abs/1512.02479.

Guido F Montufar, Razvan Pascanu, Kyunghyun Cho, and Yoshua Bengio. On the number of linear regions of deep neural networks. In *Advances in neural information processing systems*, pp. 2924–2932, 2014.

Ben Poole, Subhaneil Lahiri, Maithra Raghu, Jascha Sohl-Dickstein, and Surya Ganguli. Exponential expressivity in deep neural networks through transient chaos. In *Advances in neural information processing systems*, pp. 3360–3368, 2016.

Ramprasaath R. Selvaraju, Abhishek Das, Ramakrishna Vedantam, Michael Cogswell, Devi Parikh, and Dhruv Batra. Grad-cam: Why did you say that? visual explanations from deep networks via gradient-based localization. *CoRR*, abs/1610.02391, 2016. URL http://arxiv.org/abs/1610.02391.

L. S. Shapley. A value for n-person games. *Contributions to the Theory of Games (AM-28)*, 1(4): 169, 1953.

K. Simonyan, A. Vedaldi, and A. Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. In *Workshop at International Conference on Learning Representations*, 2014.

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

Suraj Srinivas and R Venkatesh Babu. Data-free parameter pruning for deep neural networks. *arXiv preprint arXiv:1507.06149*, 2015.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

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

Raphael Tang, Ashutosh Adhikari, and Jimmy Lin. Flops as a direct optimization objective for learning sparse neural networks. *arXiv preprint arXiv:1811.03060*, 2018.

Karen Ullrich, Edward Meeds, and Max Welling. Soft weight-sharing for neural network compression. *arXiv preprint arXiv:1702.04008*, 2017.

Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. Learning structured sparsity in deep neural networks. In *Advances in Neural Information Processing Systems*, pp. 2074–2082, 2016.

Jason Yosinski, Jeff Clune, Anh Mai Nguyen, Thomas J. Fuchs, and Hod Lipson. Understanding neural networks through deep visualization. *CoRR*, abs/1506.06579, 2015.

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

Hao Zhou, Jose M Alvarez, and Fatih Porikli. Less is more: Towards compact cnns. In *European Conference on Computer Vision*, pp. 662–677. Springer, 2016.

A APPENDIX

The bar charts visualize filter rankingse for the LeNet network with two convolutional and two fully connected layers trained on MNIST and FashionMNIST, respectively. The vertical axis describes, respectively, the Shapley value (left column) and the importance switches value (right column). The horizontal axis contains the filter indices.

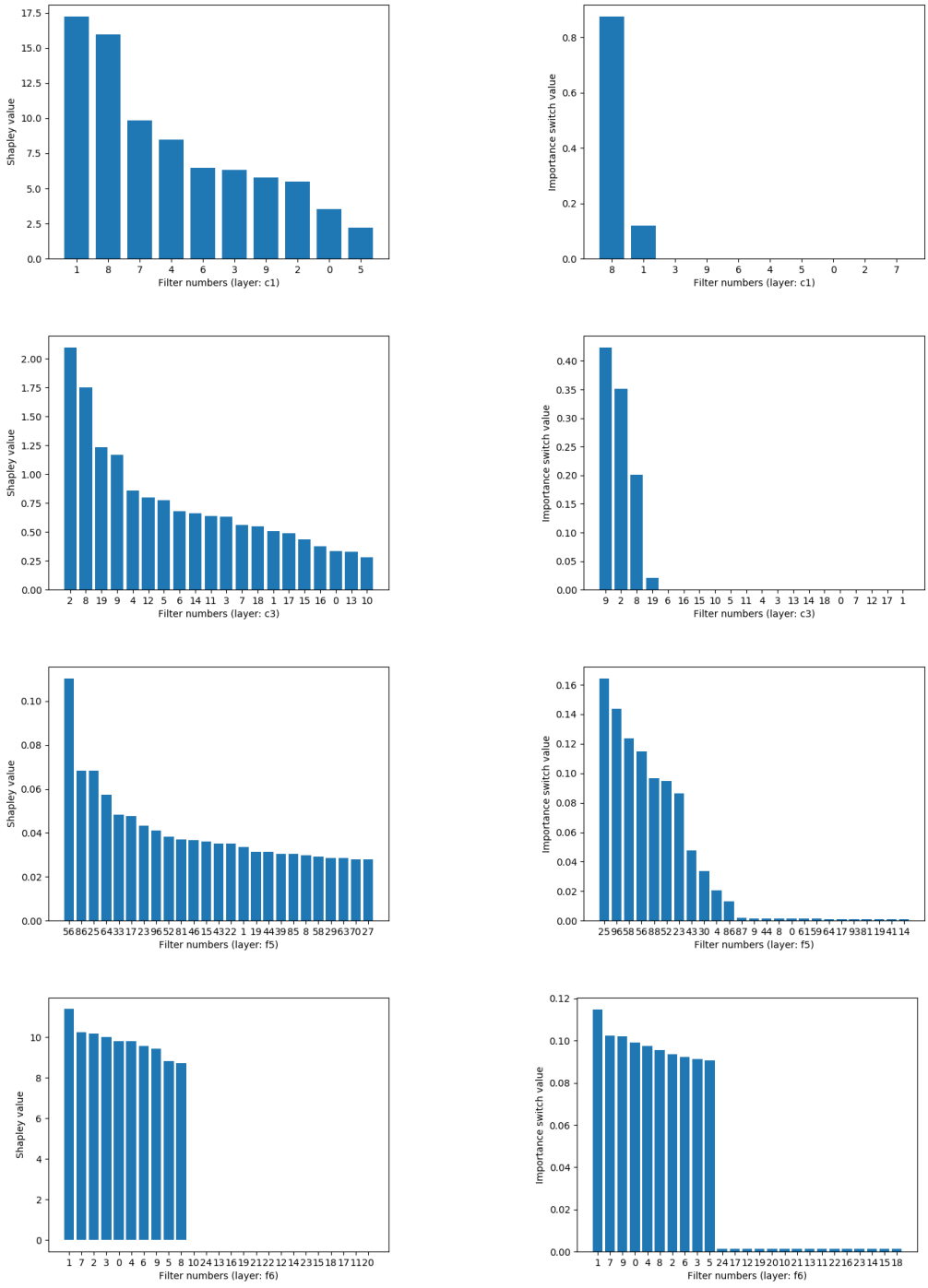

Figure 3: MNIST dataset.

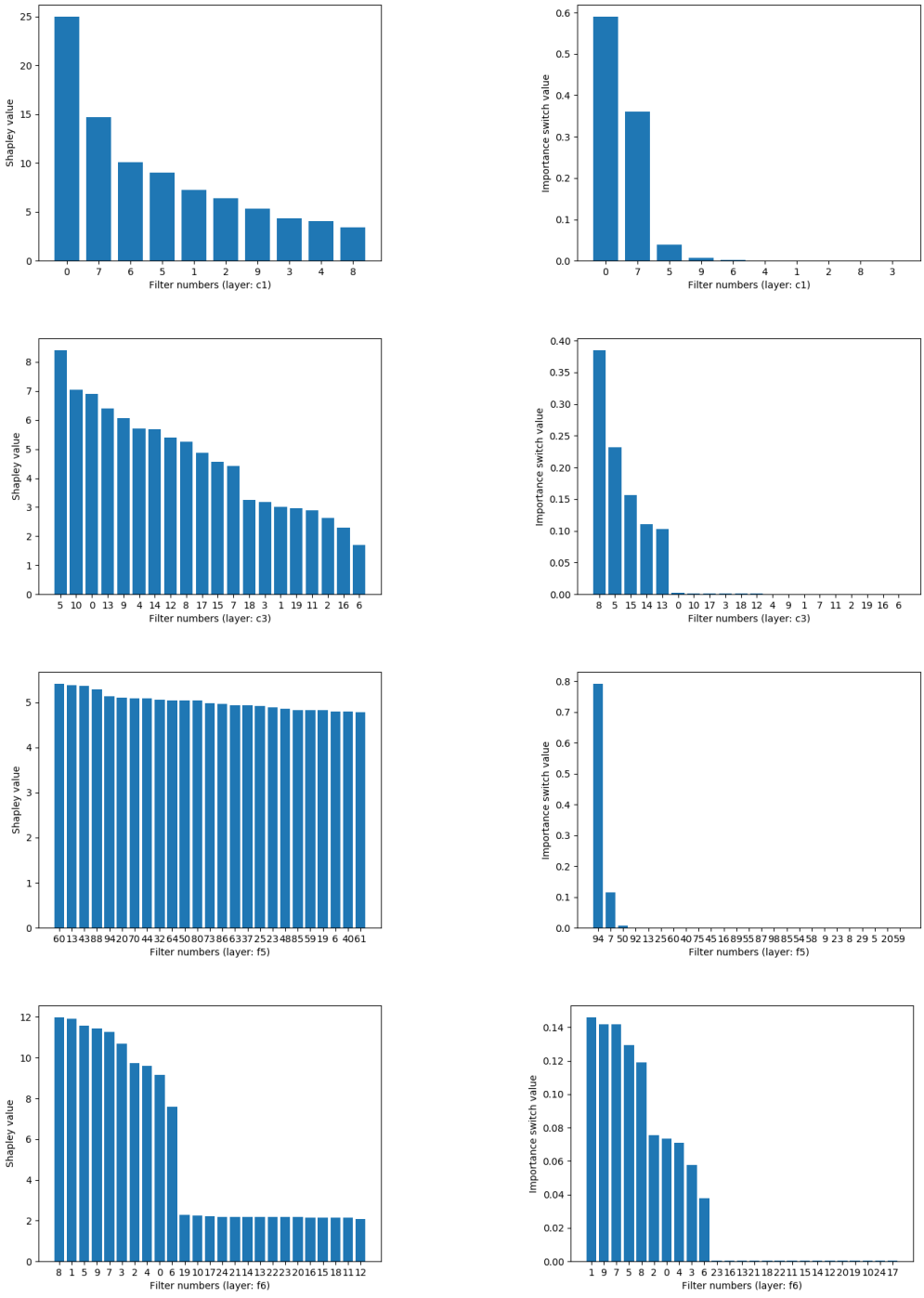

Figure 4: FashionMNIST dataset.