A SINGLE SHOT PCA-DRIVEN ANALYSIS OF NETWORK STRUCTURE TO REMOVE REDUNDANCY

Anonymous authors
Paper under double-blind review

ABSTRACT

Deep learning models have outperformed traditional methods in many fields such as natural language processing and computer vision. However, despite their tremendous success, the methods of designing optimal Convolutional Neural Networks (CNNs) are still based on heuristics or grid search. The resulting networks obtained using these techniques are often overparametrized with huge computational and memory requirements. This paper focuses on a structured, explainable approach towards optimal model design that maximizes accuracy while keeping computational costs tractable. We propose a single-shot analysis of a trained CNN that uses Principal Component Analysis (PCA) to determine the number of filters that are doing significant transformations per layer, without the need for retraining. It can be interpreted as identifying the dimensionality of the hypothesis space under consideration. The proposed technique also helps estimate an optimal number of layers by looking at the expansion of dimensions as the model gets deeper. This analysis can be used to design an optimal structure of a given network on a dataset, or help to adapt a predesigned network on a new dataset. We demonstrate these techniques by optimizing VGG and AlexNet networks on CIFAR-10, CIFAR-100 and ImageNet datasets.

1 MOTIVATION

Deep Learning is widely used in a variety of applications, but often suffers from prohibitive computational complexity due to the large number of parameters and operations involved. The design of a CNN is based on heuristics, and standard networks are often only slightly modified to adapt to newer datasets. The entire network is retrained on the new data, tuning hyperparameters such as learning rate, momentum, regularization etc. It is rare to redesign the network structure in terms of the number of layers or the number of filters per layer. But since increasing the number of parameters adds complexity to the hypothesis space under consideration, it is essential to evaluate if the increased dimensionality is indeed required for the new dataset. In fact, this analysis should be carried out for any network design, to ensure that redundancy is eliminated as much as possible within the network. Currently, there is no fixed method of arriving at a good network structure. Most designers either perform a grid search for the network architecture, or start with a variant of 8-64 filters per layer, and double the number of filters per layer as a rule of thumb. The layers are clubbed into modules, and the number of modules are increased as long as accuracy is increased [Simonyan & Zisserman (2014),He et al. (2015)]. This often results in an over-designed network, as many works on pruning show [Han et al. (2015)]. Redundancy not only increases training time and computational complexity needlessly, but also creates the need for specialized training in the form of dropout and regularization [Srivastava et al. (2014)].

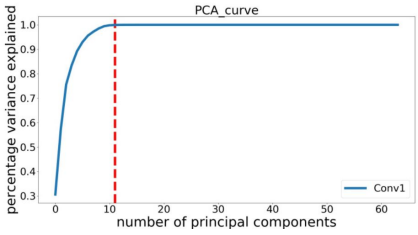
The field of model compression explores ways to prune a network post training in order to remove redundancy. It has been shown that a large percentage of the weights of standard networks can be removed without harming accuracy as long as the network is retrained [Han et al. (2015)]. However, the challenge with most pruning experiments is that they tend to be experimental, requiring large amounts of time-consuming iterations to determine the right thresholds for each layer, and then progressively retrain and proceed layer by layer. This becomes more of a search and optimization problem, which is time and compute intensive. Another challenge is that these results are not interpretable, and hence there is no way to justify or predict the results to any degree of accuracy. Our technique analyzes the network in a single pass over the network and estimates the 'significant'

dimensions of the data at each layer. The dimensions determine the number of filters to keep in that layer, and when the dimensions stop expanding with layers, there is no benefit to going deeper. The resulting networks are optimal in both width and depth, and can be found with negligible overhead of design time and compute power. Another advantage of this analysis is that it provides hardware designers a way to gracefully trade-off accuracy with computational complexity. Some applications may be more error-tolerant but power-critical, and this method exposes a knob to tune that trade-off depending on the application, without any extra iterations. It also helps understand which layers are less sensitive to pruning, and therefore can be pruned more aggressively than others.

The main contributions of the paper are explainable design heuristics for optimal network design at negligible extra compute cost or time. To the best of our knowledge, this is the first paper that analyzes all layers of networks in a single shot and identifies an optimal structure in terms of both width and depth, without any iterative searches for thresholds. For more error tolerant applications, where accuracy can sometimes be traded for faster inference or lower power consumption, this analysis provides designers a way to gracefully tune the trade-off based on their requirements. The resultant curves, which are indicative of the sensitivity of layers, also help identify layers one can afford to be aggressive with while compressing the network. We demonstrate these techniques to design optimal structures of some well known networks in Section 4.

Price of House	# Bedrooms	Sq. Feet Area	Locality Rating	....	FeatureM
2000	3	900	1	XXX	XXX
1900	2	800	1	XXX	XXX
1500	1	500	2	XXX	XXX
.	..	..	..	XXX	XXX
.	..	..	.	XXX	XXX
1800	2	750	3	XXX	XXX

(a)



(b)

Figure 1: (1a) Input to PCA is a NxM sized matrix, where M is the number of features among which we want to identify redundancy and N is the number of samples we have feature values for. (1b) A sample curve for PCA. On the X axis are the number of principal components and on the Y axis are the cumulative percentage of the variance of the input data that the components can explain.

2 Background and Previous Work

2.1 Previous work on Pruning Weights and Filters

Our exploration of traditional approaches to model compression are broadly divided into the following categories: (i) pruning weights, (ii) pruning filters, (iii) approximation techniques, (iv) quantization and (v) others.

Pruning Weights Some of the most aggressive reductions in model sizes are achieved via pruning out individual weights. LeCun et al. (1990), Hassibi et al. (1993), Han et al. (2015) all find different metrics to rank weights. However, these methods require a large number of iterations to find the thresholds and result in an unstructured sparsity in the network, which limits the actual saving in hardware.

Pruning Filters These are the reduction techniques that are most closely related to our work. It is a rapidly developing field of research that introduces structured sparsity in the network that hardware implementations can immediately leverage. Li et al. (2016) take a similar approach as Han et al., but instead of pruning individual weights, they prune out filters using the sum of magnitude of weights. Similarly, Molchanov et al. (2016) also come up with different metrics to determine saliency or significance of a filter. Both these approaches result in a good compression with minimal loss in accuracy,but the thresholds for some layers for pruning can only be found iteratively, incurring a heavy computation and retraining cost. Ding et al. (2018) automate the approach to decide threshold per layer, using an L1 norm metric but still suffer from the lack of interpretability and iterative retraining process for different layers. RoyChowdhury et al. (2018) use cosine similarity between filters as a metric to determine redundancy, and don't retrain the network. However, they notice a drop in accuracy for CNNs, and the threshold is still found by trial and error.

Approximation techniques Many papers such as Iandola et al. (2016),Howard et al. (2017), Zhang et al. (2015), Denton et al. (2014) and Jaderberg et al. (2014) use sound statistical techniques to approximate the weight matrices and make computation efficient. However, these techniques also work layer by layer and therefore need multiple iterations of retraining, which can be a significant bottleneck with large networks.
Quantization techniques Another way to prune networks is to leverage their inherent error resilience by quantizing and sharing weights. In extreme cases these weights can be quantized all the way down to 1 bit, such as in Courbariaux et al. (2016) or Rastegari et al. (2016). These methods are sometimes too aggressive and cause a degradation in accuracy for larger CNNs. Weight sharing as per Han et al. (2015) is a form of vector quantization that can be tuned to work with any network, at the cost of multiple iterations.
Other Techniques Luo et al. (2017) pose pruning as a subset selection problem based on the next layer's statistic. This approach suffers from many of the same limitations of thresholds found via trial and error and iterative retraining requirements. Yu et al. (2017) defines an importance score at the last but one layer and backpropagates it to all neurons. Given a fixed ratio, it removes the weights with scores below that. While this does get a holistic view of all layers, here too, the ratio is found empirically. Ioannou et al. (2015) have a similar idea, they learn a set of basis filters by introducing a new initialization technique. However, they decide the structure before training, and the method finds a compressed representation for that structure. In comparison our work takes a trained network, and finds the optimal structure. Hinton et al. (2015) works a little differently than the other model compression techniques. It creates a new smaller model and trains it on the outputs of the larger model. However, there is no gaurantee that the smaller model itself is optimal. Another difference is that none of these methods determine an optimal depth for the given trained network under consideration. Many of these techniques are orthogonal to this work and can be used in conjunction.

2.2 Principal Component Analysis (PCA)

PCA is a dimensionality reduction technique that takes many samples of correlated input features as input, and outputs orthogonal features that are linear combinations of the input features. The output features are ranked by the percentage of variance of data they can explain. As an analogy, consider a regression problem to predict house rates with N samples of houses and M features in each sample. The input to PCA would be an NxM sized matrix, as shown in Figure 1a. In a contrived example, consider the case of two features: number of bedrooms and square feet area. One can assume that these two will be correlated and if engineered down to one feature, such as 0.3*sq feet area + 0.7*number of bedrooms, that one feature alone may be able to capture most of the variance of the data, reducing the dimensionality of the data from 2 to 1. A sample output of PCA is shown in Figure 1b, with cumulative explained variance sketched against the number of principal components. The red line shows that the 99.9% of the total variance is explained by only 11 out of 64 features, and exposes redundancy in our feature space.

3 Identifying and Removing Redundancy

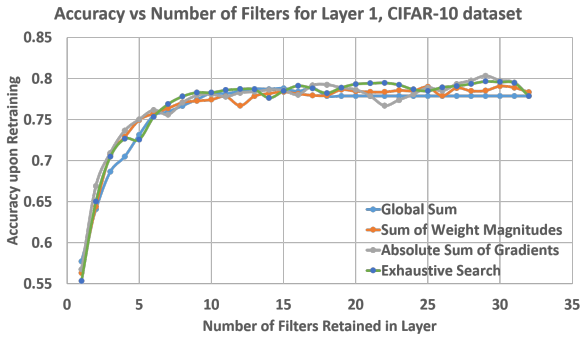
3.1 Framing Optimal Network Design as a Dimensionality Reduction Problem

The success of currently adopted pruning techniques can be attributed to the redundancy present in the network. Figure 2b shows the filters of the first layer of AlexNet. Many filters within the layer are highly correlated and potentially detect the same feature, therefore making insignificant contributions to accuracy. We aim to identify the independent filters, or the dimensionality of the space of filters we are interested in. Inspired by works on model compression, but striving for more intuitive and explainable metrics for determining 'significance' of a filter. Some metrics we used to sort and rank filters are:

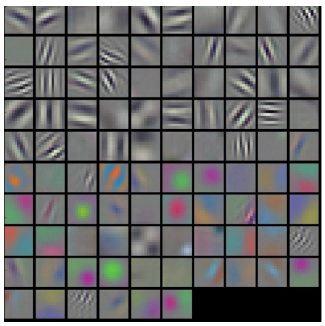
1. Deep Compression based metric: Extending Han et al.'s paper to whole filters rather than just individual weights, the absolute magnitude of all the weights making up a filter are summed and used as a measure of 'significance'. This metric is used in Li et al. (2016)

as well. However, it considers each weight as independent of others and fails to capture patterns detected by weights acting in conjunction with each other.

2. Absolute sum of gradients: This metric ranks filters based on the sum of absolute values of error gradients at each weight over the entire training set. It has a direct correlation with the contribution of each weight to the total loss, and if a weight contributes highly to loss, it has a major contribution in the output of the CNN.
3. Global summing of filter activation maps: The activation maps produced by convolving filters over an input are summed over the entire validation set. For one filter, this relates to how many times and how strongly the filter was found in the input. This is done for all filters and results in a vector that essentially represents the bag of words histogram. The intuition behind this is that if a filter is found strongly at many locations in the input, we would prefer to retain it.
4. Exhaustive search: To give a good baseline, an exhaustive search was carried out in every iteration. All the remaining filters were removed one at a time, and the net was retrained. Removal of whichever filter caused the least drop in accuracy was identified as the least significant filter for that iteration. This analysis could only be carried out since the network and the dataset both were small in size for these experiments.



(a) Result of different filter pruning metrics



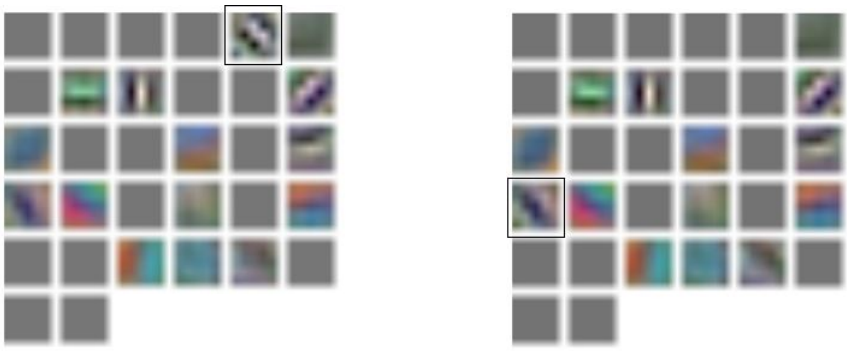
(b) Filters from the first layer of AlexNet

Figure 2

We ranked the filters based on the metrics mentioned above, and iteratively removed the least significant filter and recorded the retrained accuracy. The results comparing retraining accuracy across the metrics are shown in Figure 2 for a convolutional layer with 32 5x5 filters on CIFAR 10. Two significant observations were made: 32 filters could be reduced down to 14 with no loss in accuracy, and all metrics gave very similar results. To understand what was happening, we visualized what happens when a filter is removed and the model retrained iteratively in an animation for the first layer filters. The resulting GIF can be seen at `github_link_removed_for_anonymity`

From the GIF it appears that one or more of the remaining filters 'absorb' the characteristics of the filter that is pruned out. A particular example is shown and explained in Figure 3. The filter to be pruned out is shown on the left, and the filter that changes after retraining is on the right. It can be seen that the checkerboard pattern of the filter that was pruned out gets pronounced in the filter highlighted in Figure 3b upon retraining. Before retraining, this filter looked a little similar to the filter being pruned, but the similarity gets more pronounced upon retraining.. This pattern is repeated often in the animation, and leads to the hypothesis that as a filter is removed, it seems to be recreated in some other filter(s) that visually appear to be correlated to it. Since the accuracy did not degrade, it appears that if the network layer consists of similar or correlated filters, the network can recreate the significant filters with any of them upon retraining.

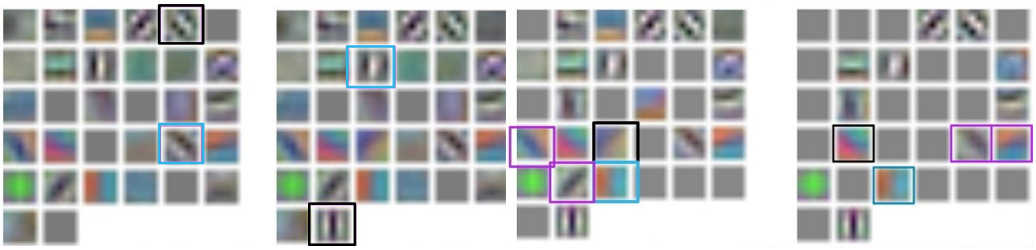
Although all metrics identified different filters to prune out, as we removed the least significant filter one by one and retrained iteratively, the number of filters removed when the accuracy of the system started to degrade was approximately the same across all metrics. This result led us to believe that the number of significant filters is an intrinsic property of a particular network structure trained on a dataset rather than dependent on some metric. We now wanted to predict the retrained filter that would absorb the pruned out filter. We tried simple techniques to quantify correlation such as pearson correlation coefficients to judge similarity between filters. We checked if the filter that changed



(a) (b)

Figure 3: Visualization of pruning. The filter to be pruned out is highlighted on the left and the filter it merges with is highlighted on the right. It can be seen that the merged filter incorporates the diagonal checkerboard pattern from the removed filter.

the most (in an L2 sense) upon pruning out a particular filter and retraining the system was the one which had the maximum pearson correlation coefficient with the filter being pruned out. Our experiments showed that this was not always true, as sometimes one filter was absorbed by multiple filters combining to give the same feature as the pruned out filter, although each of them had low correlation coefficients individually. Two examples of matches and mismatches are explained in Figure 4.



(a) (b)

Figure 4: (4a) Filter to be pruned is shown in black and the one that got changed the most is in blue. The filter in blue also had the highest pearson correlation coefficient with the filter in black. (4b) Mismatches are shown here. The filter that is pruned out is in black, the one closest to it according to pearson coefficient is in blue and the two filters that changed the most after retraining are in pink.

Rather than framing this as a subset selection problem as many pruning papers tend to intrinsically lean toward, this motivated us to view this as a 'sub-space' of the entire filter space we are interested in. We want to find the dimensionality of that space. Viewing this as a dimensionality reduction problem allows us to use PCA to determine the number of 'principal filters', and reduce the dimensionality of the space that the data resides in after passing through a layer. PCA creates new uncorrelated filters which are linear combinations of the original correlated filters. These new filters are created to capture the maximum variation of data, and are ranked according the percentage of the variance they can explain. This gives us an idea of how many filters are doing useful transformations. If there are M filters within which we want to find redundancy, the input to PCA is an NxM sized matrix where N is the number of samples for M features. PCA will output new M filters each of which will be a linear combination of all M input filters.

$$Filter'_1 = C_{11} * Filter_1 + C_{12} * Filter_2 + ... + C_{1M} * Filter_M$$
$$Filter'_2 = C_{21} * Filter_1 + C_{22} * Filter_2 + ... + C_{2M} * Filter_M$$

and so on. The original filters are lost, but each new filter contains information from all of them. Filter_1 explains the maximum variance in the data, followed by Filter_2 and so on. The exact percentage of the variance that they explain is also available to us via PCA. We want to find the least number of filters that can explain the variance of the data well. For instance, if less than M filters capture 99.9% of the variance, we don't need to have all M filters in that layer. In most cases, we are more

interested in how many filters explain some percentage variance in the data rather than the exact transformations given by these coefficients C_{ij}, as those will be learned themselves by the network upon retraining. For the purpose of this paper, we refer to the dimensions that explain 99.9% of the variance of the data as the significant dimensions.

3.2 What is a feature value for a filter?

A feature value of the number of bedrooms is 3 or 4, of square feet is 400 or 900, but it is not immediately clear what a 'feature value' of a filter might be. We need to have samples of features acting upon the same input in a way that captures redundancy between them, so the natural candidate easily available to us is the output of a convolution of an input patch with each filter. Each pixel in the output activation map created by a filter (pointed out in Figure 5) can be viewed as the feature value for that filter. Flattening the output feature map conveniently gives us many samples for that filter. Doing this across different filters gives us many samples of all filters reacting to the same input. As shown in Figure 5, activations at one location, across all channels, of the output activation map gives us one sample. Flattening the entire output activation map, we get H_o*W_o samples for all M filters and can run PCA on this directly. We then observe how many filters account for a significant amount of the total variance, typically 99.9%. We can create a new network with that many filters and retrain from scratch, or we can use the transformation matrix to change the filters into the principal filters we identified, perform the same change on the biases and on the channel dimension of the next layer, and use that as a starting point to fine tune the network. In our experiments, training from scratch gave us the desired accuracy, so in the interest of avoiding multiple iterations, we decided to skip using the transformations. However, there is one preliminary case that we examined in which the transformations are useful, which we will further discussed (Section 6).

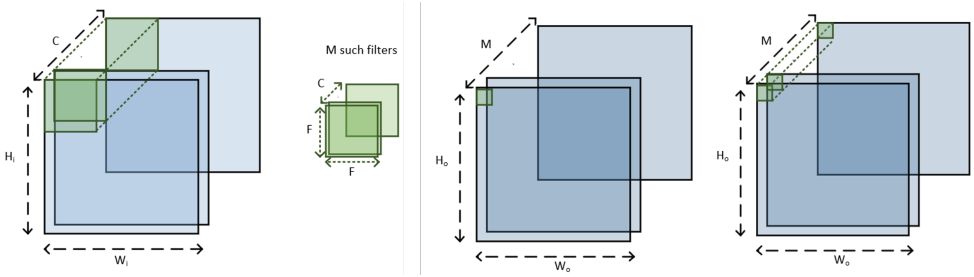


Figure 5: The output of convolving one filter with an input patch can be viewed as the feature value of that filter. In the centre, the highlighted green pixel is the feature value for the first filter for the first input patch. On the right, the same input patch creates feature values for all M filters, resulting in 1 row of the PCA matrix.

3.3 Finding an Optimal Depth

Classification using CNNs can be viewed as non linear transformations into progressively higher dimension spaces, until the data is somewhat separable. This take on neural networks leads us to hypothesize that if the number of principal filters stop expanding as we go deeper into the network, those layers are not causing an increase in the complexity of the boundary we are drawing. Experiments on adapting deep networks such as VGG-16/19 (with batch normalization) on smaller datasets like CIFAR-10/100 (Section 4) show that the principal dimensions tend to expand as you go deeper, but only until a certain point and then they begin to contract. We removed the last layer one by one and trained the system from scratch, and observed that the network does not lose accuracy until we start removing the layers in which the dimensions were still expanding. This leads to a good heuristic as to how deep a network should be, and comes without the need for any additional analysis.

3.4 How Many Data Points are Enough for PCA?

There is a trade-off between accurately representing the entire dataset and the complexity of running PCA. Ideally we would like to collect the activations over the entire dataset and run PCA on that, but the cost of running PCA on such a large dataset quickly becomes prohibitive. In our experiments,

we used randomized images from the training set, and varied the number of images over which we collected activations. We ran our PCA analysis on these collected activations to analyze when the number of filters suggested for explaining 99.9% variance stabilize. We noticed that as long as the number of samples was roughly more than 2 orders of magnitude larger than the number of filters we were trying to deduce correlation between, the results were fairly stable across different simulations. This is easy to achieve in the initial layers, as each image gives a big activation map with $H_o * W_o$ amount of samples per input image. But in the later layers, as the number of filters multiply, and the output height and width are divided due to maxpool layers, the number of images we need to collect activations over increase significantly. Still, the time taken to run PCA in all our experiments was a negligible fraction of the training time.

3.5 Sensitivity of Layers to Pruning

The shape of the PCA curves also informs us of how sensitive a layer is to compression. If the curve is very sharp, then it can be pruned more aggressively compared to a smoother curve where the principal components are well spread out with each component contributing to accuracy. This is shown in Figure 7, plotted for training VGG16_BN (with batch normalization) adapted to CIFAR-10 data. The PCA curves give an idea of how sharply the accuracy will degrade in a layer, which decides how aggressive we can afford to be while pruning a layer. In Figure 6, the first row corresponds to layers 1 through 7, and the second row corresponds to layers 8 onwards. the first column shows PCA curves, which get lesser steep as we go deeper until layer 7, and then the trend reverses. The second column shows the percentage of retained filters vs percentage accuracy upon retraining. A similar trend is seen here, as we go deeper until layer 7, an equal drop in filters results in gradually increasing drop in accuracy, and this trend reverses from layer 8 onwards. This expansion of significant dimensions until layer 7 and subsequent contraction from layer 8 leads us to hypothesize that the first 7 layers are optimal, and we can do away with the subsequent layers. This hypothesis is confirmed in the results section (Figure 8), where we see negligible accuracy degradation for removal of layers 8 through 13. The last layer before the classifier has a slightly lesser steep curve, presumably because it is directly connected to the classifier. A similar trend was observed for VGG19 on CIFAR-100 data (section 4.2).

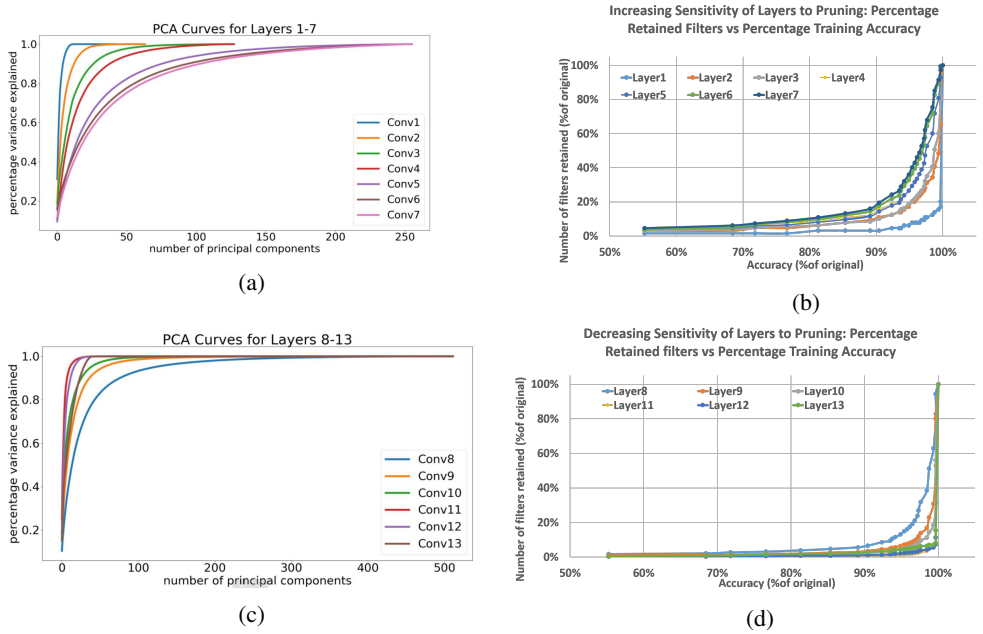


Figure 6: Curves for VGG16_BN network trained on CIFAR-10. 6a shows that layers 1-7 (6a & 6b) have increasing sensitivity to pruning, whereas the sensitivity decreased from layers 8 onwards (6c 6d). 6a and 6c show the PCA curves for explained variance vs number of principal components and 6b and 6d show percentage number of filters vs percentage of original accuracy for different layers

4 RESULTS AND OPTIMAL NETWORK STRUCTURE

The results are tabulated and summarized in Table 1. The details of the corresponding networks and experiments are explained in the following sections. A toolkit [Burzawa (2018)] available with PyTorch [Paszke et al. (2017)] was used for profiling networks to get the number of operations and parameters.

4.1 NETWORK 1: VGG16 ADAPTED TO CIFAR-10

The first network we analyzed was the VGG16 (with batch normalization) network, adapted to CIFAR-10 dataset. The configuration is shown as a vector, where each element is either the number of filters in a convolutional layer, or 'M' if it's a MaxPool layer. The network has 5 modules, each ending with a maxpool layer, and with 2 or 3 convolutional layers in each module. The initial configuration was [64, 64, 'M', 128, 128, 'M', 256, 256, 256, 'M', 512, 512, 512, 'M', 512, 512, 512]. We ran our PCA analysis, and the number of filters required to explain 99.9% variance in each layer are: [11, 42, 'M', 103, 118, 'M', 238, 249, 249, 'M', 424, 271, 160, 'M', 36, 38, 42]. Scratch-training a new network with this configuration resulted in an accuracy drop of 0.31%. More details can be found in Table 1. Having found an optimal width per layer, we analyze the resultant configuration for optimizing depth. The dimensions expand for 7 layers, and then start contracting. As discuseed in section 3.3, we claim that the optimal network design should have only 7 layers. To corroborate this, we carry out a naive analysis removing the last layer every iteration and training from scratch, for both the original and the reduced-width network and the same trend can be observed in both (Figure 7a and 7b). There is less than 0.25% drop as long as we have more than 7 layers, supporting our intuition that expansion of dimensions is indicative of a good optimal depth for a network. The number of parameters and operations have decreased by 2.8x and 7.7x respectively. The degradation curve with number of filters is shown in Figure 7b, giving designers a knob to gracefully trade-off accuracy with computational complexity.

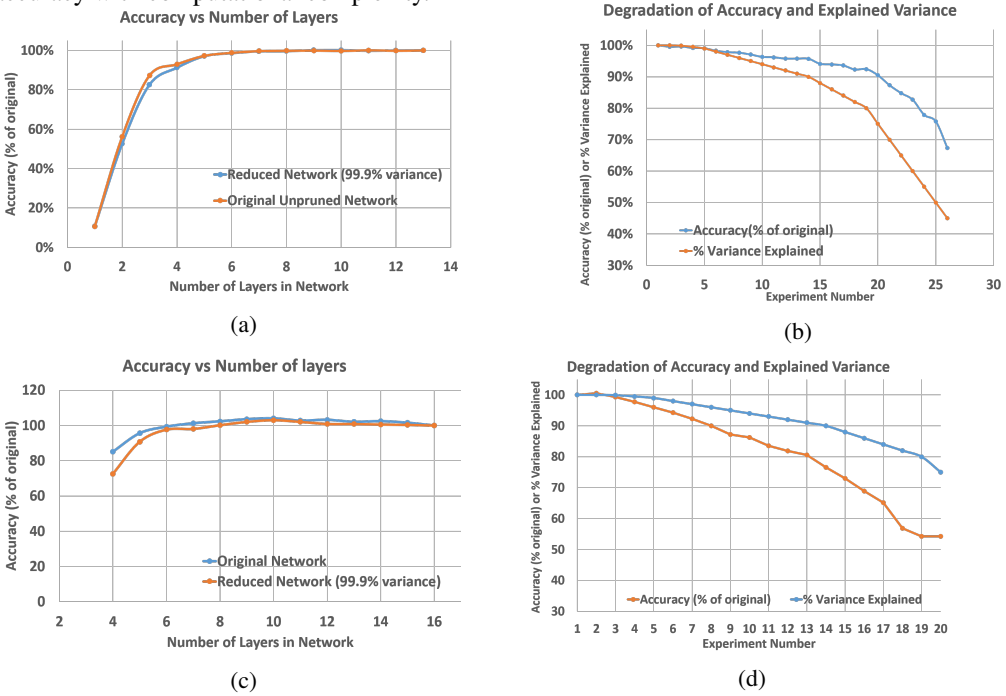


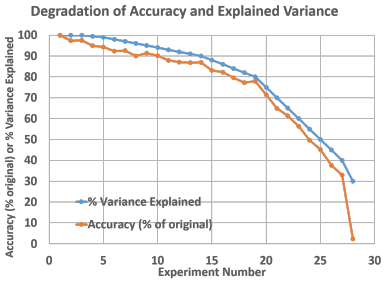
Figure 7: 7a and 7b show results for VGG16 trained on CIFAR-10 and 7c and 7d for VGG19 trained on CIFAR-100. 7a and 7c illustrate how decreasing the number of layers affects accuracy for the original network and the network with only significant filters. 7b and 7d reflect the explained variance targeted in blue and the corresponding accuracy as percentage of original accuracy in orange.

4.2 NETWORK 2: VGG19 ADAPTED TO CIFAR-100

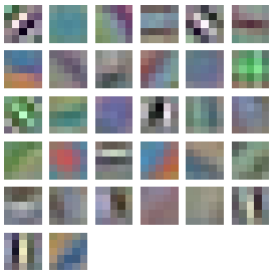
The original VGG19_BN network for CIFAR-100 dataset has 5 modules, each ending with a max-pool layer, and with 2 or 4 convolutional layers in each module. The original configuration has 16 convolutional layers with the following configuration [64, 64, 'M', 128, 128, 'M', 256, 256, 256, 256, 'M', 512, 512, 512, 512, 'M', 512, 512, 512, 512, 'M']. Analyzing the principal components of the activations of each layer for retaining 99.9% variance leads to the following configuration: [11, 45, 'M', 97, 114, 'M', 231, 241, 245, 242, 'M', 473, 388, 146, 92, 'M', 31, 39, 42, 212, 'M']. Training a new network with this configuration led to a 0.5% drop in accuracy, from 72.09% to 71.59%. To estimate an optimal depth, we notice that the first two modules are indeed expanding dimensions, but the third and fourth module each seem to have only one important layer. The last module appears to be redundant from the point of view of expansion of dimensions. Based on this analysis, we design a new network with the following structure: [11, 45, 'M', 97, 114, 'M', 231, 245, 'M', 473, 'M'] and train it from scratch. The curve for naive layerwise degradation for the original and reduce-width network is shown in Figure 7, supporting the conclusion that we can reduce the depth all the way down to 7 layers. The optimized configuration using all the same hyperparameters has 73.03% accuracy, which is approximately an improvement of 1% over the original network, presumably due to reduced overfitting. This results in a reduction of 3.8X in number of MACs and 9.1X in number of parameters. The degradation curve with number of filters is shown in Figure 7d.

4.3 NETWORK 3: ALEXNET ADAPTED TO CIFAR-100

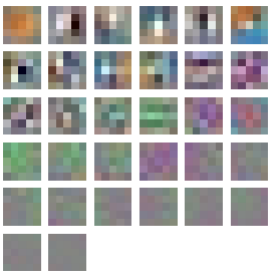
To analyze a smaller network, we trained a version of AlexNet on CIFAR-100. We used the network with the following configuration for the 5 convolutional layers: [64, 192, 384, 256, 256]. Upon doing the PCA analysis, we identified the following configuration for retaining 99.9% variance: [44, 119, 304, 251, 230] which resulted in an accuracy drop of 1% upon retraining with the reduced number of filters. Since the last layer was not expanding the significant dimensions, we removed it and scratch trained a new network of configuration of [44,124,285,477]. This gave us a further drop of 0.08% in accuracy and an improvement of 2.1X in both number of operations and parameters. The degradation curve with number of filters is shown in Figure 8.



(a)



(b)



(c)

Figure 8: (8a) For AlexNet network on CIFAR100, the curve in blue reflects the explained variance targeted and the curve in orange reflects the corresponding accuracy as percentage of original accuracy. (8b) Learned filters of the a convolutional layer with 32 filters on CIFAR-10. (8c) Orthogonal filters transformed according to principal components, ranked according to the percentage of variance of the output activation map they explain

4.4 NETWORK 4: VGG19 ADAPTED TO IMAGENET

Carrying out a similar analysis for VGG-19 (with batch normalization) for ImageNet led to the following configuration after PCA analysis: [6, 30, 'M', 49, 100, 'M', 169, 189, 205, 210, 'M', 400, 455, 480, 490, 'M', 492, 492, 492, 492, 'M']. The accuracy dropped by around 0.25% with an improvement of 1.72X and 1.07X in number of operations and parameters respectively.

	CIFAR10, VGG16_BN			
	Configuration	Accuracy	#Ops	#Params
Initial	[64, 64, 'M', 128, 128, 'M', 256, 256, 256, 'M', 512, 512, 512, 'M', 512, 512, 512]	94.07%	1X	1X
Width-Reduced	[11, 42, 'M', 103, 118, 'M', 238, 249, 249, 'M', 424, 271, 160, 'M', 36, 38, 42]	93.76%	0.53X	0.27X
Final	[11, 42, 'M', 103, 118, 'M', 238, 249, 'M', 424, 'M']	93.36%	0.35X	0.13X
	CIFAR100, VGG19_BN			
	Configuration	Accuracy	#Ops	#Params
Initial	[64, 64, 'M', 128, 128, 'M', 256, 256, 256, 256, 'M', 512, 512, 512, 512, 'M', 512, 512, 512, 512, 'M']	72.09%	1X	1X
Width-Reduced	[11, 45, 'M', 97, 114, 'M', 231, 241, 245, 242, 'M', 473, 388, 146, 92, 'M', 31, 39, 42, 212, 'M']	71.59%	0.53X	0.27X
Final	[11, 45, 'M', 97, 114, 'M', 231, 245, 'M', 473, 'M']	73.03%	0.26X	0.11X
	CIFAR100, AlexNet			
	Configuration	Accuracy	#Ops	#Params
Initial	[64, 192, 384, 256, 256]	42.77%	1X	1X
Initial-Expanded	[44,119,304,251,230]	41.74%	0.62X	0.68X
Final	[44,119,304,251]	41.66%	0.48X	0.47X
	ImageNet, VGG19_BN			
	Configuration	Accuracy	#Ops	#Params
Initial	[64, 64, 'M', 128, 128, 'M', 256, 256, 256, 256, 'M', 512, 512, 512, 512, 'M', 512, 512, 512, 512, 'M']	74.24%	1X	1X
Width- Reduced	[6, 30, 'M', 49, 100, 'M', 169, 189, 205, 210, 'M', 400, 455, 480, 490, 'M', 492, 492, 492, 492, 'M']	74.00%	0.58X	0.94X

Table 1: Summary of Results

5 CONCLUSION

This analysis has only been done on activation outputs for convolutional layers before the application of non-linearities such as ReLU. Non-linearities introduce more dimensions, but those are not a function of the number of filters in a layer. Hence we recommend not to perform ReLU in-place while performing this analysis. The number of samples to be taken into account for PCA are recommended to be around 2 orders of magnitudes more than the number of filters we are trying to find redundancy in. This is particularly of importance in the later layers where the activation map sizes are small. We need to collect these activations over many batches to make sure we have enough data to run PCA analysis on. While the percentage variance one would like to retain depends on the application and acceptable error tolerance, empirically we have found that preserving 99.9% puts us at a sweet spot for most cases with less than half a percentage point in accuracy degradation and a considerable gain in computational cost. This analysis comes in handy in three cases: While designing new network for new data; while adapting given network for new data; and while optimizing current network for faster runtimes or reduced power consumption during training or inference in hardware implementations. Another benefit of this analysis is that not only does it deliver an optimal point, but enables an interpretable, graceful exploration of accuracy-energy trade-off with negligible overhead of compute cost and time. This method is orthogonal to other model compression techniques.

6 SCOPE OF FUTURE WORK

PCA not only gives us the dimensionality, it also gives us the principal components that transform the filters into the orthogonal 'principal filters' of a layer. A visualization of this is shown in figure 9 for the first layer of a small network trained on CIFAR-10. The 32 5x5 filters learned by the layer are shown on the left. The orthogonal transformed filters are shown on the right, and they are ranked according to the percentage of variance they explain. The filters are no longer interpretable, but each is a linear combination of all 32 filters on the left. The first 16 explain 99.9% of the variance, and we can use this transformation matrix to transform the filters and biases for this layer, and the number of channels for all the filters of the next layer and use that as an initialization of a smaller network rather than training from scratch. We did not see a siginificant benefit from this in our simulations as all our networks were performing well by initializing from scratch, except when we were trying to train FCN on PASCAL VOC data. In that case, we need to initialize the encoder with VGG-16 pretrained on ImageNet or the FCN network does not learn. Preliminary analysis on FCN shows that rather than scratch-training a reduced VGG-16 on ImageNet and using that as a pretrained encoder, we can use these transformations to change the weights of that and the next layer as a good initialization. We fine-tuned for a few iterations per layer and found that the FCN was able to learn. A detailed analysis of this technique for segmentation architectures is part of future work, since the focus of this paper is on a single-shot, iteration free analysis.

REFERENCES

Lukasz Burzawa. Pytorch toolbox. https://github.com/e-lab/pytorch-toolbox, 2018.

Matthieu Courbariaux, Itay Hubara, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1. *arXiv preprint arXiv:1602.02830*, 2016.

Emily L Denton, Wojciech Zaremba, Joan Bruna, Yann LeCun, and Rob Fergus. Exploiting linear structure within convolutional networks for efficient evaluation. In *Advances in neural information processing systems*, pp. 1269–1277, 2014.

Xiaohan Ding, Guiguang Ding, Jungong Han, and Sheng Tang. Auto-balanced filter pruning for efficient convolutional neural networks. In *AAAI*, 2018.

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, David G. Stork, Gregory Wolff, and Takahiro Watanabe. Optimal brain surgeon: Extensions and performance comparisons. In *Proceedings of the 6th International Conference on Neural Information Processing Systems*, NIPS'93, pp. 263–270, San Francisco, CA, USA, 1993. Morgan Kaufmann Publishers Inc. URL http://dl.acm.org/citation.cfm?id=2987189.2987223.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015. URL http://arxiv.org/abs/1512.03385.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *CoRR*, abs/1704.04861, 2017. URL http://arxiv.org/abs/1704.04861.

Forrest N. Iandola, Matthew W. Moskewicz, Khalid Ashraf, Song Han, William J. Dally, and Kurt Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and <1mb model size. *CoRR*, abs/1602.07360, 2016. URL http://arxiv.org/abs/1602.07360.

Yani Ioannou, Duncan Robertson, Jamie Shotton, Roberto Cipolla, and Antonio Criminisi. Training cnns with low-rank filters for efficient image classification. *arXiv preprint arXiv:1511.06744*, 2015.

Max Jaderberg, Andrea Vedaldi, and Andrew Zisserman. Speeding up convolutional neural networks with low rank expansions. *arXiv preprint arXiv:1405.3866*, 2014.

Yann LeCun, John S Denker, and Sara A Solla. Optimal brain damage. In *Advances in neural information processing systems*, pp. 598–605, 1990.

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

Jian-Hao Luo, Jianxin Wu, and Weiyao Lin. Thinet: A filter level pruning method for deep neural network compression. *CoRR*, abs/1707.06342, 2017. URL http://arxiv.org/abs/1707.06342.

Pavlo Molchanov, Stephen Tyree, Tero Karras, Timo Aila, and Jan Kautz. Pruning convolutional neural networks for resource efficient inference. *arXiv preprint arXiv:1611.06440*, 2016.

Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.

Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In *European Conference on Computer Vision*, pp. 525–542. Springer, 2016.

Aruni RoyChowdhury, Prakhar Sharma, and Erik G. Learned-Miller. Reducing duplicate filters in deep neural networks. 2018.

K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.*, 15 (1):1929–1958, January 2014. ISSN 1532-4435. URL http://dl.acm.org/citation.cfm?id=2627435.2670313.

Ruichi Yu, Ang Li, Chun-Fu Chen, Jui-Hsin Lai, Vlad I. Morariu, Xintong Han, Mingfei Gao, Ching-Yung Lin, and Larry S. Davis. NISP: pruning networks using neuron importance score propagation. *CoRR*, abs/1711.05908, 2017. URL http://arxiv.org/abs/1711.05908.

Xiangyu Zhang, Jianhua Zou, Xiang Ming, Kaiming He, and Jian Sun. Efficient and accurate approximations of nonlinear convolutional networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1984–1992, 2015.