On Correlation of Features Extracted by Deep Neural Networks

Babajide O. Ayinde, *Student Member, IEEE*, Tamer Inanc, *Senior Member, IEEE*, and Jacek M. Zurada, *Life Fellow, IEEE*

***Abstract*—Redundancy in deep neural network (DNN) models has always been one of their most intriguing and important properties. DNNs have been shown to overparameterize, or extract a lot of redundant features. In this work, we explore the impact of size (both width and depth), activation function, and weight initialization on the susceptibility of deep neural network models to extract redundant features. To estimate the number of redundant features in each layer, all the features of a given layer are hierarchically clustered according to their relative cosine distances in feature space and a set threshold. It is shown that both network size and activation function are the two most important components that foster the tendency of DNNs to extract redundant features. The concept is illustrated using deep multilayer perceptron and convolutional neural networks on MNIST digits recognition and CIFAR-10 dataset, respectively.**

***Keywords*—Deep learning, feature correlation, feature clustering, cosine similarity, feature redundancy, deep neural networks.**

I. INTRODUCTION

DNNs have become ubiquitous in a wide range of applications ranging from computer vision [1–3] to speech recognition [4–6] and natural language processing [7], [8]. Over the past few years, the general trend has been that DNNs have grown deeper and wider, amounting to huge increase in their size. The number of parameters in DNNs is usually very large and no constraints are generally placed on the data and/or the model, hence offering possibility to learn very flexible and high-performing models [9]. However, this flexibility may hinder their scalability and practicality due to very high memory/time requirements, and may lead to extracting highly redundant parameters with risk of overfitting [10].

A number of studies have shown that a significant percentage of features extracted by DNNs are redundant [11–18]. As demonstrated in [11], a fraction of the parameters is sufficient to reconstruct the entire network by simply training on low-rank decompositions of the weight matrices. To this end, Optimal Brain Damage [19] and Optimal Brain Surgeon [20] exploit second-order derivative information of the loss function to localize unimportant parameters. HashedNets use a hash function to randomly group weights into hash buckets, so that all weights within the same hash bucket share a single parameter value for pruning purposes [21]. Redundant features have also been localized and pruned using simple thresholding mechanism [22].

Instead of localizing the redundant neurons in a fully-connected network, [23] compresses a trained model by identifying a subset of diverse neurons. Redundant feature maps are removed from a well trained network using particle filtering to select the best combination from a number of randomly generated masks [24]. The importance of features has also been ranked based on the sum of their absolute weights [25]. With the assumptions that features are co-dependent within each layer, [26] groups features in hierarchical order. Driven by feature map redundancy, [27] factorizes a layer into 3×3 and 1×1 combinations and prunes redundant feature maps.

These observations that DNNs are prone to extracting redundant features evidently suggest and reinforce our hypothesis that much of the information stored within DNN models may be redundant. In addition, large number of parameters also translates to model's tendency to overfitting, if trained on limited amount of data. Some of the problems associated with over-parameterization have been previously addressed via model compression [25], [28], removal of unnecessary weights [19], [20], and regularization [29], [30]. These heuristics for eliminating redundancy more often than not deteriorate the performance of the compressed model. The open question still remains: how to obtain best compact and efficient models that are free of redundancy?

Knowing the level of redundancy in models could be useful for two main reasons. First, inference-cost-efficient models can be built via pruning with small deterioration of prediction accuracy [22], [25]. This is important in practice because optimal architecture are unknown. However, pruning should enable smaller model to inherit knowledge from a larger model. Since learning a complex function directly by small suboptimal model might result in its poor performance, it is therefore necessary to first learn a task with model many parameters and to follow with pruning redundant and less important features [24]. This is particularly important for porting deep learning models to resource limited portable devices. Secondly, information about the level of redundancy in models can be used for feature diversification in order to optimize their performance since the adverse effect of redundancy in DNNs has been shown in [12], [31] [10], [12], [32].

B. O. Ayinde is with the Department of Electrical and Computer Engineering, University of Louisville, Louisville, KY, 40292 USA (e-mail: babajide.ayinde@louisville.edu).

T. Inanc is with the Department of Electrical and Computer Engineering, University of Louisville, Louisville, KY, 40292 USA, (e-mail: t.inanc@louisville.edu).

J. M. Zurada is with the Department of Electrical and Computer Engineering, University of Louisville, Louisville, KY, 40292 USA, and also with the Information Technology Institute, University of Social Science,Łódz 90-113, Poland (Corresponding author, e-mail: jacek.zurada@louisville.edu).

This work was supported by the NSF under grant 1641042.

The problem addressed in this work is three-fold: (i) we investigate the impact of modules of DNNs such as width and depth of the network, activation function, and parameter initialization on extraction of redundant features by DNNs (both fully-connected and convolutional), (ii) we estimated the number of redundant features by adapting hierarchical agglomerative clustering algorithm, and (iii) we show experimentation in order to obtain insight about which configurations (network size/activation function/parameter initialization) provide better performance tradeoff. The paper is structured as follows: Section II introduces the network configurations and the notation used in the paper. Section III introduces the notion of feature redundancy in DNN and its estimation. Section IV discusses the experimental designs and present the results. Finally, conclusions are drawn in the last section.

II. Network Configurations

Notations and network configurations in the paper in context of convolutional and fully-connected layer are briefly and separately highlighted below:

1) Convolutional layer:: By letting n'_l, h_l, and w_l, denote the number of channels, height and width of input of the l^{th} layer, respectively, input $x_l \in \mathbb{R}^p$ is transformed by a layer into output $x_{l+1} \in \mathbb{R}^q$, where x_{l+1} serves as the input in layer $l+1$. Since the layer is convolutional, p and q are given as $n'_l \times h_l \times w_l$ and $n'_{l+1} \times h_{l+1} \times w_{l+1}$, respectively. A convolutional layer convolves x_l with n'_{l+1} $3D$ features $\chi \in \mathbb{R}^{n'_l \times k \times k}$, resulting in n'_{l+1} output feature maps. Each $3D$ feature consists of n'_l $2D$ kernels $\zeta \in k \times k$. Unrolling and combining all features into a single kernel matrix $\mathbf{W} \in \mathbb{R}^{z \times n'_{l+1}}$ where $z = k^2 n'_l$. The ith feature in layer l is denoted by $\mathbf{w}_i^{(l)}$, i=1,...n'_l and each $\mathbf{w}_i^{(l)} \in \mathbb{R}^z$ corresponds to the i-th column of the kernel matrix $\mathbf{W}^{(l)} = [\mathbf{w}_1^{(l)}, \ ...\mathbf{w}_{n'_l}^{(l)}] \in \mathbb{R}^{z \times n'_{l+1}}$.

2) Fully-connected layer:: In the case of a fully-connected layer, p and q denote $n'_l h_l w_l \times 1$ and $n'_{l+1} \times 1$, respectively. A layer operation involves only vector-matrix multiplication with kernel matrix $\mathbf{W} \in \mathbb{R}^{z \times n'_{l+1}}$, where $z = n'_l h_l w_l$. Also for fully-connected layer, the ith feature in layer l is denoted by $\mathbf{w}_i^{(l)}$, i=1,...n'_l and each $\mathbf{w}_i^{(l)} \in \mathbb{R}^z$ corresponds to the i-th column of the kernel matrix $\mathbf{W}^{(l)} = [\mathbf{w}_1^{(l)}, \ ...\mathbf{w}_{n'_l}^{(l)}] \in \mathbb{R}^{z \times n'_{l+1}}$.

III. Estimating the number of redundant features

Correlation between two features can be computed by evaluating the cosine similarity measure between them as given in (1):

$$Cosine(\phi_1, \phi_2) = \frac{< \phi_1, \phi_2 >}{\| \phi_1 \| \| \phi_2 \|} \tag{1}$$

where $< \phi_1, \phi_2 >$ is the inner product of two arbitrary normalized feature vectors ϕ_1 and ϕ_2; $\phi_i = {}^{w_i}/\sqrt{||w_i||^2}$ and $i = 1, 2$. The similarity between two feature vectors corresponds to the correlation between them, that is, the cosine of the angle between them in feature space. Since the entries of feature vectors can take both negative and positive values, $Cosine(\phi_1, \phi_2)$ is bounded by [-1,1]. It is 1 when ϕ_1=ϕ_2 or when ϕ_1 and ϕ_2 are identical. $Cosine(\phi_1, \phi_2)$ is -1 when the two vectors are in exact opposite direction. The two feature vectors are orthogonal in weight space when $Cosine$ is 0.

The evaluation of pairwise feature similarities $\Omega^{(l)}$ for a given layer l can be vectorized to reduce the computational overhead. By letting $\mathbf{\Phi} = [\phi_1^{(l)}, \ ...\phi_{n'_l}^{(l)}] \in \mathbb{R}^{z \times n'_l}$ contain n'_l normalized feature vectors ϕ_i as columns, each with z elements corresponding to connections from layer $l-1$ to i^{th} neuron of layer l, then the pairwise feature similarities $\Omega^{(l)}$ for a given layer l is given as

$$\Omega^{(l)} = \mathbf{\Phi}^{\mathbf{T}^{(l)}} \mathbf{\Phi}^{(l)} \tag{2}$$

$\Omega^{(l)} \in \mathbb{R}^{n' \times n'}$ contains the inner products of each pair of columns i and j of $\mathbf{\Phi}^{(l)}$ in each position i,j of Ω in layer l. It is remarked that $\Omega^{(l)}$ can be used to roughly estimate the number of redundant features in layer l. In this work, we utilize a suitable agglomerative similarity testing/clustering algorithms to estimate the number of redundant features.

Based on a comparative review, a clustering approach from [33], [34] has been adapted and reformulated for this purpose. By starting with each feature vector ϕ_i as a potential cluster, agglomerative clustering is performed by merging the two most similar clusters C_a and C_b as long as the average similarity between their constituent feature vectors is above a chosen cluster similarity threshold denoted as τ [35], [36]. The pair of clusters C_a and C_b exhibits average mutual similarities as follows:

$$\begin{aligned} \overline{SIM_C}(C_a, C_b) = \frac{\sum_{\phi_i \in C_a, \phi_j \in C_b} Cosine(\phi_i, \phi_j)}{|C_a| \times |C_b|} > \tau \\ a, b = 1, ...n'_l; \ a \neq b; \ \ i = 1, ...|C_a|; \\ j = 1, ...|C_b|; \quad and \quad i \neq j \end{aligned} \tag{3}$$

It must be noted that the definition of similarity in (3) uses the graph-based-group-average technique, which defines cluster proximity/similarity as the average of pairwise similarities of all pairs of features from different clusters. This work also considers other similarity definitions such as the single and complete links. Single link defines cluster similarity as the similarity between the two closest feature vectors that are in different clusters. On the other hand, complete link assumes that cluster proximity is the proximity between the two farthest feature vectors of different clusters. In this work, experiments based on average proximity were reported because of their superior performance. It is strongly believe that graph-based-group-average performs better because all cluster members contributed in the decision making process.

The objective of the clustering algorithm is to discover n_f features in the set of n' original weight vectors (or simply features), where $n_f \leq n'$. Upon detecting these distinct n_f clusters, a representative feature from each of these n_f clusters is randomly sampled without replacement and the remaining features in that cluster are tagged as redundant. The number

of redundant features in a particular layer l is then estimated as in (4).

$$n_r^{(l)} = n_l' - n_f^{(l)} \tag{4}$$

To illustrate the impact of activation functions and weight initializations on DNNs' susceptibility to extracting redundant features, we considered five popular activation functions (namely: Sigmoid, Tanh, ReLU, ELU, and SeLU) and five weight initializations. The activation functions considered are briefly highlighted below. Given any finite dimensional vector $\mathbf{z}$ for which activation function is to be computed for, the following four activation functions are defined as follows:

1) Sigmoid:
$$\sigma(\mathbf{z}) = \frac{1}{1+e^{-\mathbf{z}}} \tag{5}$$
2) Hyperbolic Tangent:
$$Tanh(\mathbf{z}) = \frac{e^{\mathbf{z}} - e^{-\mathbf{z}}}{e^{\mathbf{z}} + e^{-\mathbf{z}}} \tag{6}$$
3) Rectified Linear Units [37]:
$$ReLU(\mathbf{z}) = \max(0, \mathbf{z}) \tag{7}$$
4) Exponential Linear Units [38]:
$$ELU(\mathbf{z}) = \begin{cases} \mathbf{z} & \mathbf{z} > 0 \\ \alpha(e^{\mathbf{z}} - 1) & ||\mathbf{z}|| \leq 0 \end{cases} \tag{8}$$
where α is an hyperparameter that controls the value at which ELU activation function saturates for inputs with negative values.
5) Scaled Exponential Linear Units [39]:
$$SeLU(\mathbf{z}) = \lambda \begin{cases} \mathbf{z} & \mathbf{z} > 0 \\ \alpha e^{\mathbf{z}} - \alpha & ||\mathbf{z}|| \leq 0 \end{cases} \tag{9}$$
where $\lambda > 1$ and α are derived from the input. SeLU also uses a custom weight initialization with zero mean and standard deviation of $\sqrt{1/\text{size of input vector}}$.

Also, the five popular weight initialization heuristics considered are briefly described as follows:

1) random_uniform: initializes weights between -0.05 and 0.05 from a uniform distribution
2) orthogonal [40]: initializes weight through the generation of orthogonal matrix with scalar gain factor g (chosen to be 1.0 in our experiments), where gain is a multiplicative factor that scales the orthogonal matrix
3) xavier [41]: is also known as Xavier uniform initialization and it initializes weights within [-κ κ] from uniform distribution where κ is given as:
$$\kappa = \sqrt{\frac{6}{n_l' + n_{l+1}'}} \tag{10}$$
where n_l' is the number of input units and n_{l+1}' is the number of output units.
4) he_normal [42]: initializes weight with samples drawn from a truncated normal distribution centered around 0 with standard deviation of $\sqrt{\frac{2}{n_l'}}$
5) lecun_normal [43]: initializes weight with samples drawn from a truncated normal distribution centered around 0 with standard deviation of $\sqrt{\frac{1}{n_l'}}$.

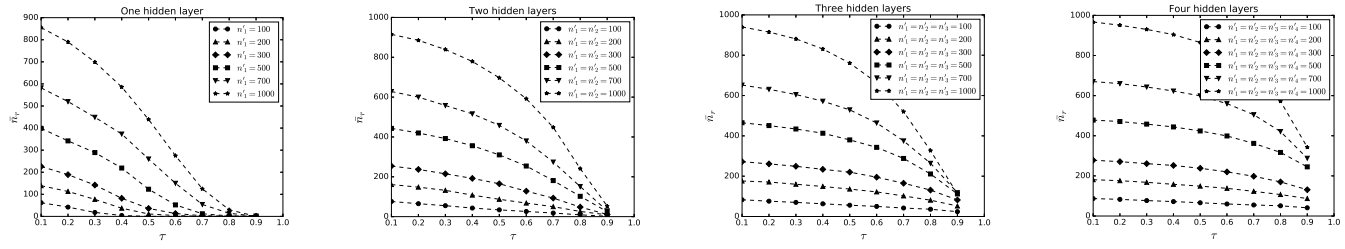
IV. Experiments

In the first set of experiments we considered a fully-connected network trained and evaluated on MNIST dataset of handwritten digits. All experiments were performed on Intel(r) Core(TM) i7-6700 CPU @ 3.40Ghz and a 64GB of RAM running a 64-bit Ubuntu 14.04 edition. The software implementation has been in Pytorch library [1] on two Titan X 12GB GPUs and the feature clustering was implemented in SciPy ecosystem [44]. The standard MNIST dataset has 60000 training and 10000 testing examples. Each example is a grayscale image of an handwritten digit scaled and centered in a 28 $\times$ 28 pixel box. Adam optimizer [45] with batch size of 128 was used to train the model for 200 epochs.

The number of redundant features was computed as in (4) after the models have been fully trained. Figures 1 a,b,c, and d show the performance of multilayer perceptron with one, two, three, and four hidden layer(s), respectively. The average number of redundant features across all layers of the network is denoted as $\bar{n}_r$. It can be observed in Figure 1 that both width (number of hidden units per layer) and depth (number of layers in the network) increase $\bar{n}_r$. As the number of hidden units per layer increases, $\bar{n}_r$ grows almost linearly. Also, the higher the number of hidden layers in a network, the higher the average number of redundant features extracted and the higher the average feature pairwise correlations. For instance, the network with one hidden layer and 100 hidden units does not have any feature pair correlated above 0.4. However, as the depth increases (for two or more hidden layers) more feature pairs have correlation above 0.4. This observation is similar for other hidden layer sizes (200, 300, 500, 700, and 1000) and depth. In particular, as can be observed in Figure 1d that many feature pairs in deep multilayer network (with four hidden layers) are almost perfectly correlated with cosine similarity of 0.9 even with just 100 hidden units per layer.

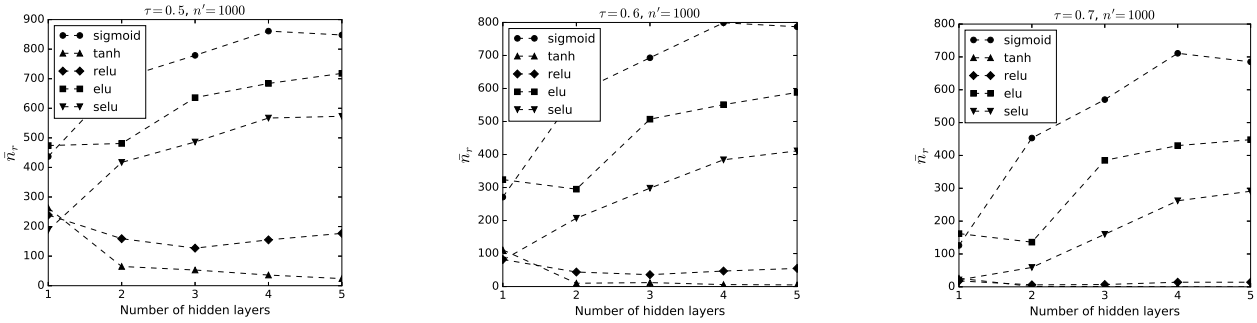
In the second set of experiments, we also used the MNIST dataset to see the impact of activation function and number of layers in DNNs on susceptibility to redundant feature extraction. We considered five popular activation functions namely: *Sigmoid*, *Tanh*, *ReLU*, *ELU*, and *SeLU*. In order to focus on the effect of activation function and number of layers, we fixed the width (number of hidden units) of the network for all layers and was set to 1000. For all networks, the weights were initialized randomly by sampling from normal distribution with zero mean and standard deviation of 0.01. Pairwise feature similarity was measured at thresholds $\tau = 0.5$, 0.6, and 0.7. As shown in Figure 2 for all thresholds, *sigmoid*, *ELU*, and

[1] http://pytorch.org/



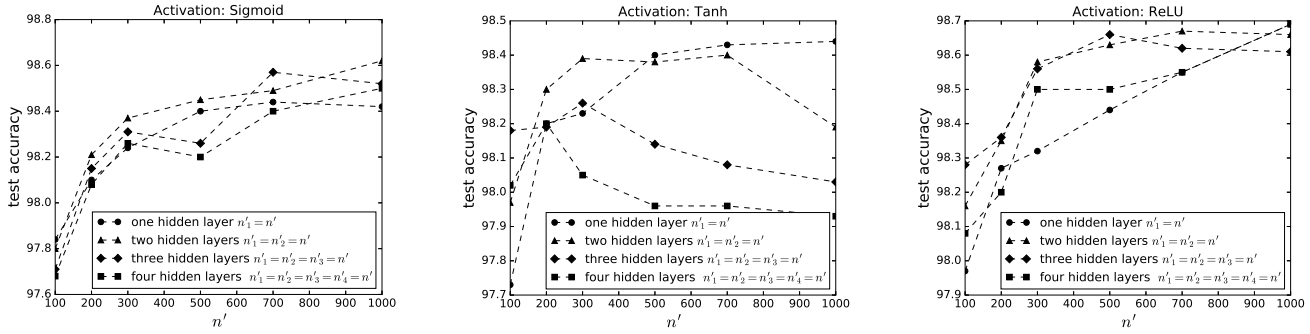
(a) (b) (c) (d)

Fig. 1: Average number of redundant features across all layers ($\bar{n}_r$) vs threshold τ with (a) one (b) two (c) three, and (d) four hidden layers of multilayer percerptron using MNIST dataset. Network width is the number of hidden units per layer and network depth is the number of hidden layers. Networks with more than one hidden layer have equal number of hidden units in all layers.



(a) (b) (c)

Fig. 2: Average number of redundant features vs number of hidden layers of multilayer percerptron for four activation functions with (a) $\tau = 0.5$ (b) $\tau = 0.6$ (c) $\tau = 0.7$ using MNIST dataset. Number of hidden units (n') per layer is 1000



(a) (b) (c)

Fig. 3: Performance on test set of MNIST dataset using (a) Sigmoid (b) Tanh (c) ReLU activation function

SeLU have higher tendency to extract redundant features than those of *Tanh* and *ReLU*. In fact, it was observed that the average number of redundant feature extracted for *ReLU* and *Tanh* did not increase as the number of layers increased. In fact, as can be observed in Figure 2b, the redundancy slightly decreases for both *ReLU* and *Tanh* as opposed to *Sigmoid*, *ELU*, and *SeLU* where it is almost always increasing.

Also, Figure 3c reinforces the observation that ReLU activation is able to outperform both sigmoid and tanh activations in Figures 3a and b for very deep networks and exhibits inherent nature to extract more diverse features. It can be observed in Figures 3 that *ReLU* benefits from both width and depth than its counterparts. As width and depth increase, the performance of tanh deteriorates while that of sigmoid heavily fluctuates. Deep multilayer network was also evaluated based on the distribution of data in high level feature space.

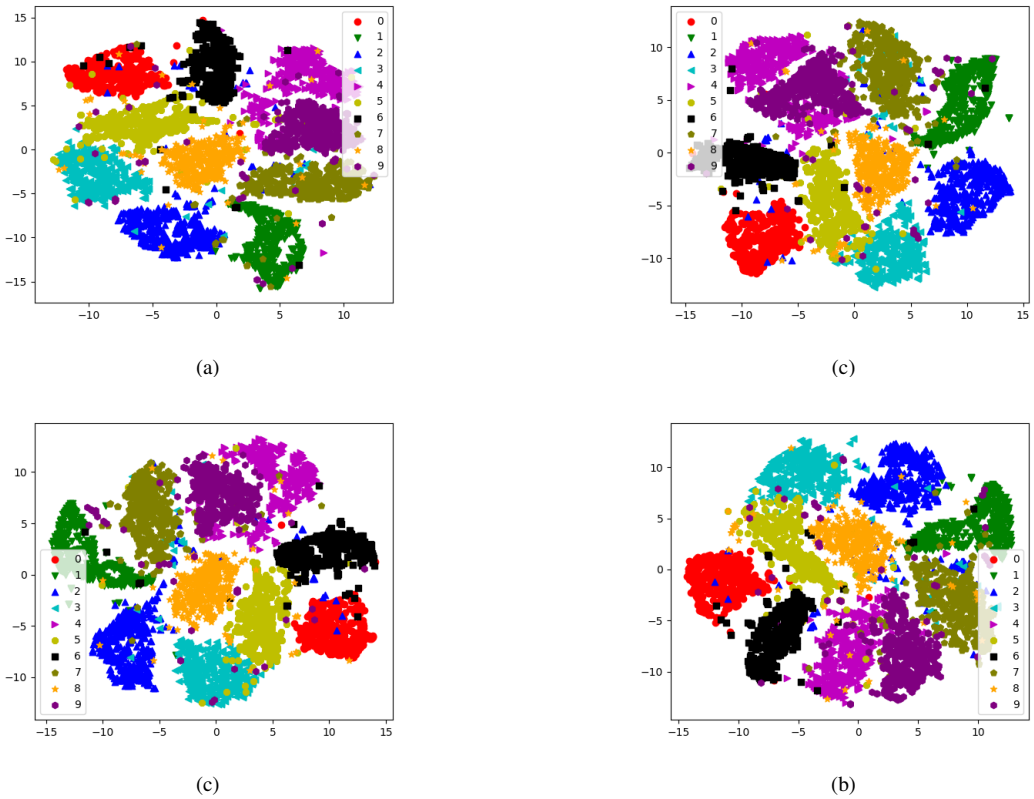


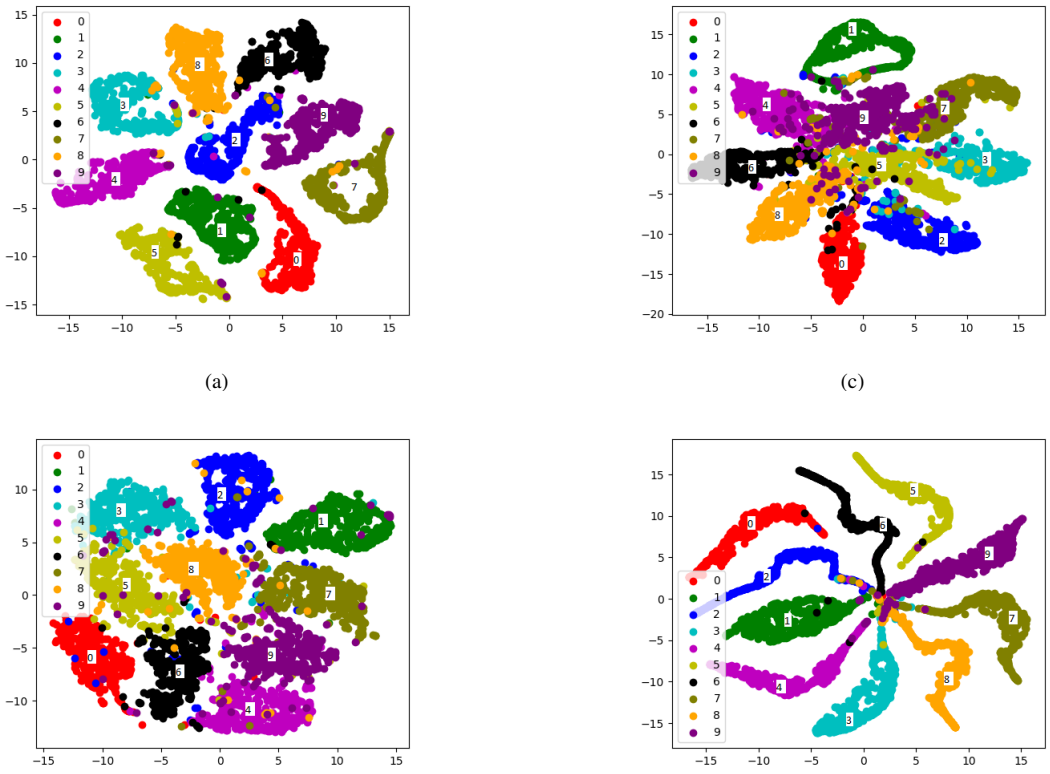
Fig. 4: t-SNE projection [46] of the hidden activation of multilayer perceptron using (a) Sigmoid (b) SeLU (c) Tanh (d) ReLU activation function trained on 5000 MNIST handwritten digits test samples. All Networks have 1000 hidden units

In this regard, t-distributed stochastic neighbor embedding (t-SNE) [46] was used to project and visualize the last hidden activations of single-hidden-layer and four-hidden-layer networks using Sigmoid, SeLU, Tanh, and ReLU activations into 2D. The projections of single layer networks and that of four-layer networks are as shown in Figures 4 and 5, respectively. The t-SNE projections show that networks with four hidden layers have clustered activations compared to that of a single layer resulting in within class holes. This is observation is pronounced for Sigmoid and SeLU activations.

In the third set of experiments on MNIST, five weight initialization heuristics were tested and the width of the network per layer was also fixed and set to 1000. Sigmoid was used in this set of experiments and only the initialization method and number of layers were varied. As shown in Figure 6, all weight initializations for shallow networks have similar tendency to extract redundant features. As the number of layers increases, however, *he_normal* [42] extracts less redundant features than all its counterparts. This observation is relatively consistent for thresholds $\tau = 0.5$, 0.6, and 0.7 as shown in Figures 6a, b, and c, respectively. This might explain why it usually outperforms other initialization methods in most vision task.

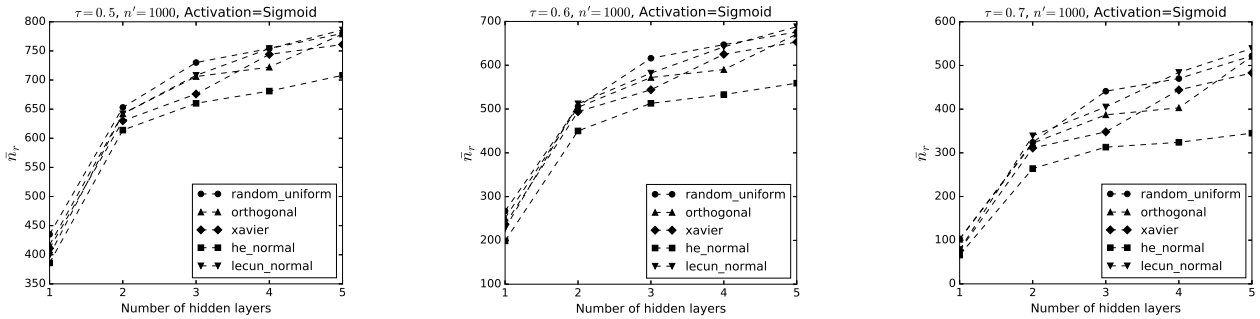
In the last set of experiments, we trained deep convolutional neural networks (VGG-11,13,16,19) on CIFAR-10 dataset to see how depth is impacting redundant feature extraction. VGG architecture [1] is a high capacity network designed originally for ImageNet dataset. We used a modified version of the VGG network architecture, which has c convolutional layers and 2 fully connected layer. Constant c in VGG-11, VGG-13, VGG-16, and VGG-19 are 8, 10, 13, and 16, respectively. In the modified version of VGG architectures, each layer of convolution is followed by a Batch Normalization layer [47]. CIFAR-10 dataset contains a labeled set of 60,000 32x32 color images belonging to 10 classes: airplanes, automobiles, birds, cats, deer, dogs, frogs, horses, ships, and trucks. The dataset is split into 50000 and 10000 training and testing sets, respectively. Our baseline model was trained for 300 epochs, with a batch-size of 128 and a learning rate 0.1. The learning rate was reduced by a factor of 10 at 150 and 250 epochs.

As shown in Table I, the average number of redundant fetaures across all layers ($\bar{n}_r$) also increases as number of convolutional layers increases. It can be observed that with 13 layers of convolution, the performance of the model starts deteriorating and the percentage of redundant feature increases by more than 21% for all τ values considered. Another crucial observation is that networks (VGG-11 and VGG-13) with 8 and 10 convolutional layers have relatively similar level of redundancy, especially for $\tau = 0.7$. This means, in



(a) (c) (c) (b)

Fig. 5: t-SNE projection [46] of the hidden activation of last layer of 4-layer perceptron network using (a) Sigmoid (b) SeLU (c) Tanh (d) ReLU activation function trained on 5000 MNIST handwritten digits test samples. All Networks have 1000 hidden units in all layers



(a) (b) (c)

Fig. 6: Average number of redundant features vs number of hidden layers of multilayer perceptron for five weight initialization methods with (a) $\tau = 0.5$ (b) $\tau = 0.6$ (c) $\tau = 0.7$ using MNIST dataset. Number of hidden units (n') per layer is 1000 and sigmoid activation is used for all layers.

relative terms, that both VGG-11 and VGG-13 have smaller degree of overfitting compared to VGG-16 and VGG-19 as reflected in their test accuracies. This may suggest a strong correlation between the level of redundancy in a model and its generalization. It must be noted that the test error of VGG-11 is higher than its counterparts; we believe that perhaps VGG-11 with 8 layers of convolution is somehow underfitting the CIFAR-10 dataset.

TABLE I: Performance of VGG models on Cifar-10 dataset. τ is the threshold of similarity.

VGG		$\bar{n}_r$ (%)				
Model	# Conv Layers	$\tau = 0.4$	$\tau = 0.5$	$\tau = 0.6$	$\tau = 0.7$	test accuracy (%)
VGG-11	8	34.0	24.1	17.7	12.8	92.09
VGG-13	10	37.8	26.9	19.1	12.9	93.65
VGG-16	13	58.8	52.6	45.5	37.2	93.51
VGG-19	16	72.4	68.3	64.5	60.3	93.24

V. Conclusion

This paper shows how size, choice of activation function, and weight initialization impact redundant feature extraction of deep neural network models. The number of redundant features is estimated by agglomerating features in weight space according to a well-defined similarity measure. Experiments were carried out using benchmark datasets and select models. The results show that both width and depth strongly correlate with redundant feature extraction. It is also established that the wider and deeper a network becomes, the higher is its tendency to extract redundant features. It has also been empirically shown on select examples that *ReLU* activation function enforces extraction of less redundant features in comparison with other activations function considered. Also, the *he_normal* initialization heuristic presented in [42] offers the advantange of extracting more distinct features for deep networks than other popular initialization heuristics considered. We illustrated the concept using fully-connected and convolutional neural networks on MNIST handwritten digits and CIFAR-10.

References

[1] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *ICLR*, 2015, pp. 1–14.

[2] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems*, 2012, pp. 1097–1105.

[3] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 770–778.

[4] A. Graves, A.-r. Mohamed, and G. Hinton, "Speech recognition with deep recurrent neural networks," in *IEEE International Conference on Acoustics, Speech and Signal Processing*, 2013, pp. 6645–6649.

[5] A. Graves and N. Jaitly, "Towards end-to-end speech recognition with recurrent neural networks," in *Proc. of the 31st International Conference on Machine Learning*, 2014, pp. 1764–1772.

[6] A. v. d. Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, and K. Kavukcuoglu, "Wavenet: A generative model for raw audio," *arXiv preprint arXiv:1609.03499*, 2016.

[7] R. Jozefowicz, O. Vinyals, M. Schuster, N. Shazeer, and Y. Wu, "Exploring the limits of language modeling," *arXiv preprint arXiv:1602.02410*, 2016.

[8] N. Shazeer, A. Mirhoseini, K. Maziarz, A. Davis, Q. Le, G. Hinton, and J. Dean, "Outrageously large neural networks: The sparsely-gated mixture-of-experts layer," *arXiv preprint arXiv:1701.06538*, 2017.

[9] C. Liu, Z. Zhang, and D. Wang, "Pruning deep neural networks by optimal brain damage," in *Fifteenth Annual Conference of the International Speech Communication Association*, 2014, pp. 1092–1095.

[10] J. Yoon and S. J. Hwang, "Combined group and exclusive sparsity for deep neural networks," in *International Conference on Machine Learning*, 2017, pp. 3958–3966.

[11] M. Denil, B. Shakibi, L. Dinh, N. de Freitas *et al.*, "Predicting parameters in deep learning," in *Advances in Neural Information Processing Systems*, 2013, pp. 2148–2156.

[12] P. Rodríguez, J. Gonzàlez, G. Cucurull, J. M. Gonfaus, and X. Roca, "Regularizing cnns with locally constrained decorrelations," in *arXiv preprint arXiv:1611.01967*, 2016, pp. 1–11.

[13] Y. Bengio and J. S. Bergstra, "Slow, decorrelated features for pretraining complex cell-like networks," in *Advances in Neural Information Processing Systems*, 2009, pp. 99–107.

[14] S. Changpinyo, M. Sandler, and A. Zhmoginov, "The power of sparsity in convolutional neural networks," *arXiv preprint arXiv:1702.06257*, 2017.

[15] B. O. Ayinde and J. M. Zurada, "Nonredundant sparse feature extraction using autoencoders with receptive fields clustering," *Neural Networks*, vol. 93, pp. 99–109, 2017.

[16] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *arXiv preprint arXiv:1510.00149*, 2015.

[17] S. Han, J. Pool, S. Narang, H. Mao, E. Gong, S. Tang, E. Elsen, P. Vajda, M. Paluri, J. Tran *et al.*, "Dsd: Dense-sparse-dense training for deep neural networks," in *ICLR*, 2017, pp. 1–13.

[18] B. O. Ayinde and J. M. Zurada, "Clustering of receptive fields in autoencoders," in *Neural Networks (IJCNN), 2016 International Joint Conference on*. IEEE, 2016, pp. 1310–1317.

[19] Y. LeCun, J. S. Denker, and S. A. Solla, "Optimal brain damage," in *Advances in Neural Information Processing Systems 2*, D. S. Touretzky, Ed. Morgan-Kaufmann, 1990, pp. 598–605. [Online]. Available: http://papers.nips.cc/paper/250-optimal-brain-damage.pdf

[20] B. Hassibi and D. G. Stork, "Second order derivatives for network pruning: Optimal brain surgeon," in *Advances in Neural Information Processing Systems*, 1993, pp. 164–171.

[21] W. Chen, J. Wilson, S. Tyree, K. Weinberger, and Y. Chen, "Compressing neural networks with the hashing trick," in *International Conference on Machine Learning*, 2015, pp. 2285–2294.

[22] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," in *Advances in Neural Information Processing Systems*, 2015, pp. 1135–1143.

[23] Z. Mariet and S. Sra, "Diversity networks," in *ICLR*, 2016, pp. 1135–1143.

[24] S. Anwar, K. Hwang, and W. Sung, "Structured pruning of deep convolutional neural networks," *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, vol. 13, no. 3, p. 32, 2017.

[25] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, "Pruning filters for efficient convnets," in *ICLR*, 2017, pp. 1–12.

[26] Y. Ioannou, D. Robertson, R. Cipolla, and A. Criminisi, "Deep roots: Improving cnn efficiency with hierarchical filter groups," *arXiv preprint arXiv:1605.06489*, 2016.

[27] X. Zhang, J. Zou, K. He, and J. Sun, "Accelerating very deep convolutional networks for classification and detection," *IEEE Transactions on*

Pattern Analysis and Machine Intelligence, vol. 38, no. 10, pp. 1943–1955, 2016.

[28] A. Polyak and L. Wolf, "Channel-level acceleration of deep face representations," *IEEE Access*, vol. 3, pp. 2163–2175, 2015.

[29] H. Zhou, J. M. Alvarez, and F. Porikli, "Less is more: Towards compact cnns," in *European Conference on Computer Vision*. Springer, 2016, pp. 662–677.

[30] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li, "Learning structured sparsity in deep neural networks," in *Advances in Neural Information Processing Systems*, 2016, pp. 2074–2082.

[31] P. Xie, Y. Deng, and E. Xing, "On the generalization error bounds of neural networks under diversity-inducing mutual angular regularization," *arXiv preprint arXiv:1511.07110*, 2015.

[32] M. Cogswell, F. Ahmed, R. Girshick, L. Zitnick, and D. Batra, "Reducing overfitting in deep networks by decorrelating representations," in *ICLR*, 2016, pp. 1–12.

[33] B. Walter, K. Bala, M. Kulkarni, and K. Pingali, "Fast agglomerative clustering for rendering," in *IEEE Symposium on Interactive Ray Tracing*, 2008, pp. 81–86.

[34] C. Ding and X. He, "Cluster merging and splitting in hierarchical clustering algorithms," in *Proc. of the IEEE International Conference on Data Mining*, 2002, pp. 139–146.

[35] B. Leibe, A. Leonardis, and B. Schiele, "Combined object categorization and segmentation with an implicit shape model," in *Workshop on Statistical Learning in Computer Vision*, vol. 2, no. 5, 2004, p. 7.

[36] S. Manickam, S. D. Roth, and T. Bushman, "Intelligent and optimal normalized correlation for high-speed pattern matching," *Datacube Technical Paper*, 2000.

[37] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *Proceedings of the 27th international conference on machine learning*, 2010, pp. 807–814.

[38] D.-A. Clevert, T. Unterthiner, and S. Hochreiter, "Fast and accurate deep network learning by exponential linear units (elus)," in *ICLR*, 2016, pp. 1–14.

[39] G. Klambauer, T. Unterthiner, A. Mayr, and S. Hochreiter, "Self-normalizing neural networks," *arXiv preprint arXiv:1706.02515*, 2017.

[40] A. M. Saxe, J. L. McClelland, and S. Ganguli, "Exact solutions to the nonlinear dynamics of learning in deep linear neural networks," *arXiv preprint arXiv:1312.6120*, 2013.

[41] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, 2010, pp. 249–256.

[42] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *Proc. of the IEEE International Conference on Computer Vision and Pattern Recognition*, 2015, pp. 1026–1034.

[43] Y. LeCun, L. Bottou, G. B. Orr, and K.-R. Müller, "Efficient backprop," in *Neural networks: Tricks of the trade*. Springer, 1998, pp. 9–50.

[44] E. Jones, T. Oliphant, P. Peterson *et al.*, "SciPy: Open source scientific tools for Python," Online accessed: 01-04-2018. [Online]. Available: http://www.scipy.org/

[45] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[46] L. v. d. Maaten and G. Hinton, "Visualizing data using t-sne," *Journal of machine learning research*, vol. 9, no. Nov, pp. 2579–2605, 2008.

[47] 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.