IMPACT OF BATCH NORMALIZATION ON CONVOLUTIONAL NETWORK REPRESENTATIONS

A PREPRINT

Hermanus L. Potgieter [1,2,4] **Coenraad Mouton** [1,2] **Marelie H. Davel** [1,2,3]
[1]North-West University [2]CAIR [3]NITheCS [4]SANSA
potgieterharmen@gmail.com

January 27, 2025

ABSTRACT

Batch normalization (BatchNorm) is a popular layer normalization technique used when training deep neural networks. It has been shown to enhance the training speed and accuracy of deep learning models. However, the mechanics by which BatchNorm achieves these benefits is an active area of research, and different perspectives have been proposed. In this paper, we investigate the effect of BatchNorm on the resulting hidden representations, that is, the vectors of activation values formed as samples are processed at each hidden layer. Specifically, we consider the sparsity of these representations, as well as their implicit clustering – the creation of groups of representations that are similar to some extent. We contrast image classification models trained with and without batch normalization and highlight consistent differences observed. These findings highlight that BatchNorm's effect on representational sparsity is *not* a significant factor affecting generalization, while the representations of models trained with BatchNorm tend to show more advantageous clustering characteristics.

1 Introduction

Deep learning has become a particularly important set of machine learning techniques and is widely applied to solve real-world tasks. At the same time, many open questions remain with regard to the ability of these deep neural networks (DNNs) to generalize so well, that is, their ability to perform well on unseen data. Although there is not yet a theoretical framework to assist us in reasoning about these models [2], the generalization ability of DNNs has been studied from many perspectives, such as the geometry of the loss landscape [3], statistical measures of stability and robustness [4], size of margins (distance to the decision boundary between classes) [5], and information-theoretic techniques [6], among others.

A promising research direction is to study the characteristics of the internal data representations formed by DNNs, where each representation is the vector of activation values from a specific layer for a given sample. Aspects of these representations that have been studied include the size of margins in the representation space [7, 8, 9]; the 'quality' of representations, evaluated using the consistency of class-specific representations and their robustness when combined [9]; and representation sparsity, that is, the number of non-zero elements in a data representation [10]. In this work, we also study the characteristics of the internal representations of DNNs, but focus on the effect that a very specific technique – Batch Normalization (BatchNorm) – has on internal representation quality.

BatchNorm [11] is a popular technique used to normalize hidden activations when training DNNs. Networks trained with BatchNorm show desirable properties such as faster convergence and better generalization ability [12, 13]. Despite the success and widespread adoption of BatchNorm, the exact mechanisms by which BatchNorm achieves its performance remain unclear. In this work, we study the effect of BatchNorm on the internal representations of convolutional neural

This work is a preprint of a published paper by the same name [1]. The authenticated version is available online at https://doi.org/10.1007/978-3-031-78255-8_14

networks (CNNs). More precisely, we analyze the resulting sparsity and clustering characteristics of CNN activations for models trained with and without BatchNorm. We selected these two elements, specifically:

- Representational sparsity is generally considered a desirable property for DNNs, and is considered to be one of the reasons for the performance of rectified linear unit (RELU) activation functions [10]. Although the sparsity of neural networks parameters (parametric sparsity) is well studied, the sparsity of the internal representations (representational sparsity) is less explored [14, p. 251]. The effect of BatchNorm on representational sparsity has been studied to a limited extent for Multilayer Perceptrons (MLPs) [15] but no such analysis has yet been performed for CNNs.
- The clustering characteristics of representations have shown to be a promising avenue to better understand generalization in DNNs [7, 16, 9]. As such, comparing the clusters formed by models trained with and without BatchNorm could provide insights into how BatchNorm increases generalization performance.

In summary, the objective of this paper is to compare models trained with and without BatchNorm and to analyze the effect of BatchNorm on the hidden representations.

2 Background

This section introduces the BatchNorm technique as well as current views on the rationale for its performance. We also review current perspectives on the effect of representation sparsity and clustered representations on generalization performance.

2.1 Batch Normalization

Batch normalization [11] is a popular layer normalization technique which speeds up training and improves the performance of deep neural networks. The goal of BatchNorm is to normalize each batch of activation values for some layer to a learned mean and standard deviation during both training and inference. More precisely, for the activation values $\mathbf{x}^l$ at layer l for a sample $\mathbf{x}$, the BatchNorm operation is defined as:

$$\mathrm{BN}(\mathbf{x}^l) = \boldsymbol{\gamma} \odot \frac{\mathbf{x}^l - \hat{\boldsymbol{\mu}}^l_{\mathcal{B}}}{\hat{\boldsymbol{\sigma}}^l_{\mathcal{B}}} + \boldsymbol{\beta}. \tag{1}$$

where $\hat{\boldsymbol{\mu}}^l_{\mathcal{B}}$ and $\hat{\boldsymbol{\sigma}}^l_{\mathcal{B}}$ is the *per feature* mean and standard deviation vectors calculated over a batch of samples $\mathbf{X}_{\mathcal{B}}$. The vectors $\boldsymbol{\gamma}$ and $\boldsymbol{\beta}$ are learnable parameters of the same dimensionality as $\mathbf{x}^l$. In the case of CNNs, each channel is considered a ‘feature’, and the mean and standard deviation are calculated independently per channel. More precisely, the activation values for a convolution layer l for a batch of samples can be denoted as a 4D tensor such that $\mathbf{X}^l_{\mathcal{B}} \in \mathbb{R}^{N \times C \times H \times W}$. Here N, C, H, and W correspond to the number of samples in the batch, the number of channels, and the height and width of the representation, respectively. The per-channel mean and standard deviation are calculated in the same way but for a channel instead of a node.

Despite the widespread popularity of BatchNorm, there is not yet a consensus on the exact mechanism by which BatchNorm speeds up training and improves generalization. Originally the success of BatchNorm was motivated by its reduction of ‘internal covariance shift’ [11], meaning that BatchNorm reduces distributional differences between the inputs to different layers in the network. However, Santurkar et al. [13] show that even when the internal distributions are artificially shifted (by injecting noise after BatchNorm layers) there is no noticeable degradation in performance. Instead, they argue from an optimization perspective, and conclude that BatchNorm drastically smooths the loss landscape. From a similar perspective, Bjorck et al. [17] show that without BatchNorm, a small subset of the activations in deeper layers grow uncontrollably large during training. They further show that BatchNorm prevents this phenomenon and allows for the use of larger learning rates that result in speedier convergence. Similarly, Li and Arora [12] show that an exponential learning rate schedule is made possible by BatchNorm.

Although the optimization perspective is the dominant view, several other authors have also investigated the benefits of BatchNorm from alternative perspectives. These include the effect of BatchNorm on adversarial robustness [18], its effect on the density of linear regions in the input space [19], and the orthogonality of internal representations [20].

2.2 Sparsity

Representational sparsity refers to the extent to which a representation consists of values that are zero or close to zero [14, p. 251]. Glorot et al. [10] approach sparsity from a biological perspective, arguing the use of rectified neurons over sigmoidal neurons is inspired by neuroscience. In addition to the use of rectified neurons, they propose the use of L_1 regularization on the activation values to promote sparsity as a mechanism for better performance. Sparsity has been attributed with benefits such as information disentangling, efficient variable-size representation, and representations being more linearly separable [10]. Representational sparsity can thus be seen as desirable in CNNs [21].

Pretorius et al. [15] investigated the effect of activation functions on Multi-Layer Perceptrons (MLPs) by analysing network characteristics such as network sparsity. They found that networks trained with ReLU activation are more sparse on average than networks trained on sigmoidal activations. They also investigated the effect of BatchNorm on representation sparsity and found that the models trained with BatchNorm tend to have *less* sparse representations. It was also found that the less sparse BatchNorm models tend to have better generalization performance, and it was suggested that sparsity is not necessarily an indication of good generalization.

2.3 Clustered Representations

Clustering is an unsupervised learning method that aims to form groups of similar elements based on different metrics. Methods can range from forming clusters by minimizing or maximizing a quality measure [22], using hierarchical methods that build a dendrogram to iteratively merge or split clusters based on a distance metric [23], or using linear algebra directly (eigenvalues and eigenvectors) to cluster [24].

2.3.1 Class-based Clustering

A neural network can be conceptualized as a clustering algorithm, as it inherently groups data with similar features together within its hidden layers [25], meaning similar samples are modelled in a similar way. Since the network is trained to predict the correct class of a given sample, it tends to cluster samples from the same class together. That said, the initial layers often capture features that are shared across different classes, leading to inter-class clustering. However, as observed by Caron et al. [26], deeper layers typically contain more class-specific information, resulting in distinct clusters that align closely with the true class labels. Class-based cluster analysis therefore assigns class labels as cluster identifiers and evaluates cluster characteristics.

2.3.2 Class-Agnostic Clustering

An alternative viewpoint is not to consider the class label information at all but rather to consider the purity of clusters formed at different layers of a network. In this case, the number of clusters that are relevant is not known ahead of time.

2.3.3 Clustering Techniques

One of the most popular clustering methods is k-means clustering [22]. The method partitions the samples into k clusters. The distance to each cluster centre is calculated, and the sample is placed in the cluster with the nearest mean. After each iteration, the cluster centres are moved to the centre of the points assigned to the cluster. The process is iterated for a number of steps. Different variations of the method exist such as k-means++ [27], Fuzzy C-means clustering [28], and k-medians clustering [29], all of which use a similar principle as k-means clustering. The k-means++ algorithm improves upon k-means with a more intelligent selection of the initial centroids. Fuzzy C-means allow data points to belong to multiple clusters, assigned via the similarity to the centroid. It focuses more on the membership relationship and uncertainty of the data points. K-medians minimise the sum of absolute differences between the centroids and data points, which is a more robust approach. The number of clusters must be specified in advance for the k-means method, which often necessitates a hyperparameter search to determine the optimal number of clusters.

2.3.4 Evaluating cluster quality

The Davies-Bouldin index (DBI) [30] is a popular metric to evaluate cluster quality. It evaluates the average similarity between each cluster and its most similar cluster, offering an indication of how well-separated and cohesive the clusters are. It is computed by comparing the average distance between clusters and the size of the clusters. This method can be used to compare the relative appropriateness of different data partitions. Importantly, the index is not sensitive to the number of clusters or the specific partitioning of the data. The DBI is defined as:

$$DBI = \frac{1}{k}\sum_{i=1}^{k}\max_{i\neq j}\left\{\frac{d(x_i)+d(x_j)}{d(c_i,c_j)}\right\} \tag{2}$$

where k denotes the number of clusters, i, j are cluster labels, $d(x_i)$ and $d(x_j)$ are the mean intra-cluster distance for the clusters i and j, respectively, and $d(c_i, c_j)$ is the distance between the cluster centroids. The DBI can be used to measure the purity of the of representational clusters based on the class that the representation belongs to.

2.3.5 Clustered representations and generalization

There are limited studies that explore the link between the characteristics of representation clusters and generalization directly. The two main papers that make a direct link between these concepts are those by Natekar and Sharma [9] and Liao et al. [16].

Natekar and Sharma proposed three measures of quality of representations to predict generalization in a post-hoc diagnostic setting [9]. The first metric they measure is consistency, which is particularly relevant, as it is defined as an indication of representation cluster purity. To measure consistency, the DBI is calculated on a dimensionally reduced version of the activation values of a layer. This produces a measure of how consistent the representations are within a cluster, as well as how different representations are across clusters. Dimensionality reduction is performed with principal component analysis (PCA) and max pooling. The main observation from their study was that simple measures based on the quality of internal representations are somewhat predictive of the generalization performance and they, therefore, conjecture that clustered representations of objects in internal layers aid in generalization.

Liao et al. [16] use the process of clustering representations as a regulariser, and demonstrate some performance gains in the process. That is, they adjust the loss function during optimization to encourage cluster-forming. They looked at three different clustering methods namely: sample clustering, spatial (channel) clustering and co-clustering of both samples and channels. The regularisation term that they add to the standard loss consists of the distance of the element to the cluster mean. The cluster centres are initialised and updated with k-means, and then resampled and updated through the $k\text{-}means++$ procedure. Their results across clustering techniques were fairly similar, with sample and spatial clustering outperforming co-clustering with a small margin.

Clustering has been studied more extensively in the unsupervised learning setting. Caron et al. [26] use the clustering of internal representations to train CNNs. The training process consists of an alternating process of clustering internal representations (using standard k-means) and updating the parameters until convergence. Once the clustering is complete, the cluster tags are used as pseudo-labels for the corresponding hidden representation vectors. The pseudo-labels are used to train a classifier network on top of the CNN to predict the pseudo-labels for each hidden representation vector. This idea was further developed by Zhan et al. [31] who introduced a batch-aware version that addresses instability caused by samples changing their batch membership; and by Li et al. [32] who forced the training process to create representations that are close to the cluster centroids.

3 Experimental setup

In this section, we describe the approach used to investigate the effect of BatchNorm on CNN representations. We select a set of CNN architectures and datasets and develop comparative models. We then develop metrics of sparsity and techniques to evaluate the cluster characteristics of representations.

3.1 Models

We consider two different datasets and network architectures for experimentation. For datasets, we select MNIST [33] and CIFAR10 [34]. MNIST, while simple, is useful for probing generalization from a clustering perspective as it consists of well-separated classes. On the other hand, CIFAR10 consists of more natural images and a greater amount of overlap among classes. This allows us to verify whether results on MNIST also hold on CIFAR10. We select architectures that match the datasets: a simple CNN with 4 convolutional layers trained on the MNIST dataset, and a standard, deeper architecture with 13 convolutional layers trained on the CIFAR10 dataset. For each of these architectures, we train a model with and without BatchNorm using 4 different random initialization seeds each.

3.1.1 MNIST

For the MNIST dataset, we make use of an architecture similar to the 'standard architecture' described by Nakkiran et al. [35]. This 5-layer CNN consists of 4 convolutional layers with $[16, 32, 64, 128]$ channels, respectively. There are also max pooling layers after the 2nd and 3rd convolutional layers with a stride of 2. We alter the architecutre by adding a fully connected hidden layer of 100 nodes preceding the output layer of 10 nodes, as we are interested in the representations formed by both the convolutional and fully connected layers. This is in contrast to the architecture of Nakkiran et al. [35] which only has a single fully connected output layer of 10 nodes.

Hyperparameters are chosen based on a hyperparameter sweep that tracks validation accuracy at the point of interpolation. We apply a ReLU activation after each hidden layer, cross-entropy as loss function, and SGD as optimizer for both the BatchNorm and non-BatchNorm models. We use a batch size of 64, a learning rate of 0.01, a learning rate schedule that multiplies the learning rate with 0.99 every 10 epochs, and train for $1\ 000$ epochs. We also include a Nesterov momentum of 0.99. We terminate the optimization once the network interpolates the training data (reaches 100% training accuracy). The accuracy achieved during training is displayed in Table 1. Both sets of models interpolated on the training data with the BatchNorm models performing a bit better on the validation and test sets than the non-BatchNorm. The variation between seeds is minimal for both the BatchNorm and non-BatchNorm models.

3.1.2 CIFAR10

For the CIFAR10 dataset, we rely on the well-known VGG-16 [36] architecture. This architectures consists of 13 ReLU-activated convolutional layers with $[64, 64, 128, 128, 256, 256, 256, 512, 512, 512, 512, 512, 512]$ channels and a kernel size of 3×3. There are also 2×2 max pooling layers after the 2^{nd}, 4^{th}, 7^{th}, 10^{th}, and 13^{th} convolutional layers with a stride of 2. We do not use the hyperparameters from the original implementation of Simonyan and Zisserman [36] but rather, perform a hyperparameter sweep using the validation set. This is done to achieve as close to current benchmarks on the dataset and architecture as possible. Specifically, we make use of cross-entropy loss in combination with the ADAM optimizer. We use an initial learning rate of 0.001 for the models with BatchNorm, and a smaller learning rate of 0.0001 for those without. The learning rate is multiplied by 0.99 after each epoch, and the momentum and stability terms of ADAM are kept at the default values. We use the same batch size of 64 for both sets of models. We train for 300 epochs using early stopping with a patience of 20% on the validation error.

The accuracy for both sets of models is in Table 1. Both sets of models achieved near interpolation on the training data, with the BatchNorm models significantly outperforming the non-BatchNorm models on the unseen data from both the training and evaluation sets. Additionally, the BatchNorm models exhibited minimal variation between seeds, whereas the accuracy of the non-BatchNorm models showed much greater variability across different seeds on the unseen data.

Table 1: Train, validation, and evaluation classification accuracy for the BatchNorm (BN) and non-BatchNorm (NBN) models on MNIST and CIFAR10. $\pm$ indicates the standard deviation across 4 random initialization seeds.

	Training Acc	**Validation Acc**	**Evaluation Acc**
CNN MNIST BN	100.000 ± 0.000	99.355 ± 0.010	99.198 ± 0.100
CNN MNIST NBN	100.000 ± 0.000	99.250 ± 0.087	99.150 ± 0.072
VGG-16 CIFAR10 BN	99.988 ± 0.024	87.080 ± 0.315	85.99 ± 0.320
VGG-16 CIFAR10 NBN	99.982 ± 0.021	81.030 ± 1.008	79.435 ± 1.059

3.2 Sparsity

We only use the training data representations during analysis, as we are specifically focused on data that both models have seen and learned. This approach provides an indication of where these two models differ, given that it is data both have memorized. To extract the representations of the samples, we pass each sample through the network and record the activation values of each layer after the ReLU activation function. This produces a representation $\mathbf{x}^l \in [0, \infty]$ for each sample at each layer of the network, and results in the 4D tensor $\mathbf{X}^l_{\mathcal{B}}$ for a batch of samples, as described in Section 2.1. When analysing a certain dimension, such as the layer or channels of the network, the tensor is unfolded into a 2D matrix where the one dimension is the size of the element analysed and the other is the product of all other dimensions. For example, analysing the sample dimension gives a matrix of $\mathbf{X}^l_{\mathcal{B}} \in \mathbb{R}^{N \times CHW}$.

We define a sparse element as any element with a value of zero. While activation values close to zero could have minimal impact on the network's decision-making process and could therefore also be considered as values for which the network effectively does not activate [14, p. 251], we do not consider such small values as sparse elements. To calculate sparsity, we first identify the inactive elements – those with activation values of zero. The tensor is then reshaped to focus on the specified axis. For channel sparsity, the resulting 2D matrix is $\mathbf{X}^l_{\mathcal{B}} \in \mathbb{R}^{C \times NHW}$ and the sparsity calculated as:

$$s_c = \frac{1}{N \cdot H \cdot W} \sum_{j=1}^{NHW} \mathbb{I}(\mathbf{x}^l_{c,j} = 0) \tag{3}$$

where $\mathbf{x}^l_{c,j}$ is the element of the matrix at position (c, j) where c is the channel, and $\mathbb{I}(\cdot)$ is the indicator function, which equals 1 if the condition inside is true and 0 otherwise. To calculate the layer sparsity we use the vector

$\mathbf{x}^l_{\mathcal{B}} \in \mathbb{R}^{1\times NCHW}$ as follows:

$$s_l = \frac{1}{N \cdot C \cdot H \cdot W} \sum_{j=1}^{NCHW} \mathbb{I}(\mathbf{x}^l_j = 0) \tag{4}$$

resulting in a single sparsity value per layer (per model and seed).

3.3 Clustered Representations

The same representations ($\mathbf{x}^l$) as before are used when analysing clustering characteristics. Due to the high dimensionality of the representations and the computational complexity associated with the high dimensionality, we reduce the representations to $\mathbf{X}^l \in \mathbb{R}^{N\times C}$. This is achieved by taking the average of the elements in the feature map across the height H and width W for each channel, expressed as:

$$\mathbf{X}^l_{n,c} = \frac{1}{H \cdot W} \sum_{h=1}^{H} \sum_{w=1}^{W} \mathbf{X}^l_{n,c,h,w} \tag{5}$$

where $\mathbf{X}^l_{n,c,h,w}$ denotes the element of the tensor $\mathbf{X}^l$ at position (n, c, h, w), and $\mathbf{X}^l_{n,c} \in \mathbb{R}^{N\times C}$ is the resulting 2D tensor. Each element represents the average value of the feature map over the spatial dimensions H and W for each channel C and sample N. We then normalize each sample representation by dividing it with its Euclidean norm to produce $\mathbf{x}^\mathbf{l}$ to be $\in [0, 1]$.

We consider cluster purity from both a class-based and class-agnostic approach. For the class-based approach, we do not apply any clustering techniques to the representations, but instead, we analyze the clusters naturally formed by the network for each labelled class. In this context, we measure the DBI of the layer representations using the labels of the sample classes as the cluster labels, following the method of Natekar and Sharma [9].

For the class-agnostic approach, we first use the k-means clustering algorithm to group similar representations and provide cluster labels for each representation. To select the optimal number of clusters we first use 2 to 15 clusters during an initial clustering step. We then measure the purity of these representation clusters using the DBI metric, as discussed in Section 5.2. We then select the optimal number of clusters as the clusters that produced the lowest DBI score.

4 Sparsity Analysis and Results

In this section, we analyze and compare the sparsity of networks trained with and without BatchNorm. We consider the sparsity of representations both per channel (Section 4.1) and per layer (Section 4.2).

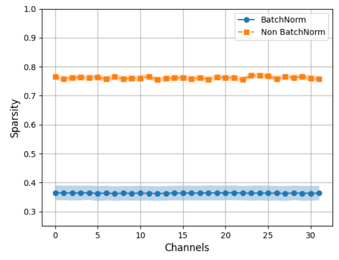
4.1 Channel Sparsity

We begin by analyzing the sparsity of individual channels within the network for selected layers. The analysis uses the representations of the hidden layers with dimensions $\mathbf{x}^l \in \mathbb{R}^{C\times NHW}$, as described in Section 3.2. The sparsity for each layer is calculated according to Equation 3, using 5 000 randomly selected training samples. We select a subset of layers spread through the networks and evaluate the same layers per model type.

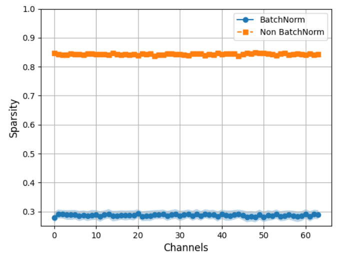
Figures 1 and 2 illustrate the average channel sparsity for the four models for both BatchNorm and Non-BatchNorm configurations, across the MNIST and CIFAR10 datasets, respectively. The standard deviation of metrics across different seeds is indicated with shading. We choose layers $6, 10$ and 13 to probe the networks in the deeper layers at consistent intervals.

For the standard CNN models trained on MNIST, as shown in Figure 1, we observe that the models trained with BatchNorm generally exhibit lower channel sparsity compared to those trained without BatchNorm. This observation is consistent across all channels and layers of the model, with the exception of the final layer, where channels in the BatchNorm models are more sparse than those in the Non-BatchNorm models. We note that channel sparsity remains consistent across different initialization seeds, and are also relatively uniform within each layer. The one exception is the last layer, where there is a slightly larger degree of variation in channel sparsity, especially for BatchNorm models.

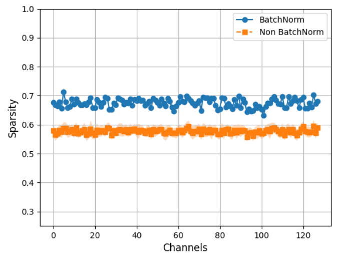
However, when we turn our attention to Figure 2, which shows the sparsity per channel for the VGG-16 models trained on CIFAR10, the opposite behaviour is observed. The VGG-16 models trained with BatchNorm tend to be more sparse than those trained without. We also note greater variation among different initialization seeds, especially for the Non-BatchNorm models. That said, we again note that each channel shows similar sparsity within a layer.



(a) Layer 2

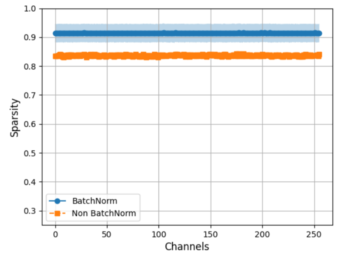


(b) Layer 3

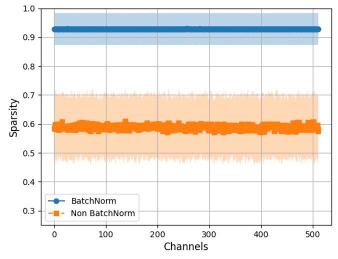


(c) Layer 4

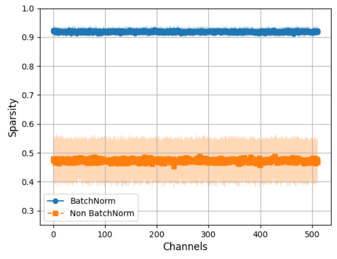
Figure 1: Average channel sparsity for BatchNorm (blue) and Non-BatchNorm (orange) models for selected convolutional layers of the standard CNN models trained on MNIST. Error bars show the standard deviation across 4 random initialization seeds.



(a) Layer 6



(b) Layer 10



(c) Layer 13

Figure 2: Average channel sparsity for BatchNorm (blue) and Non-BatchNorm (orange) models in selected VGG-16 convolutional layers trained on CIFAR10, with error bars showing the standard deviation across 4 random initialization seeds.

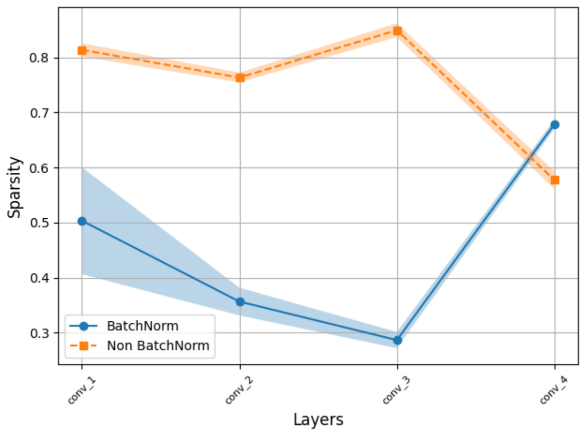
When comparing Figures 1 and 2, we observe that the channels in the CIFAR10 models trained with BatchNorm are significantly more sparse than their MNIST counterparts. This could potentially be an artefact of the different datasets, as MNIST consists of many 0-valued inputs in itself whereas CIFAR10 consists of more uniformly distributed values larger than 0. It is therefore strange that the CIFAR10 BatchNorm models are *more* sparse than those of MNIST.

4.2 Layer Sparsity

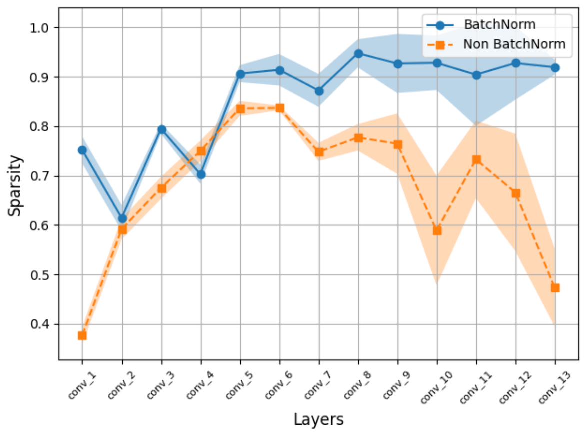
In this section, instead of analysing a single channel individually, we analyse the sparsity of each layer within the networks. This analysis uses the representations of the hidden layers with dimensions $\mathbf{x}^l \in \mathbb{R}^{N \times CHW}$, as described in Section 3.2. We once again make use of $5\,000$ randomly selected training samples. For each layer, the sparsity is computed according to Equation 4.

Figure 3 illustrates the average layer sparsity for both the MNIST (left) and CIFAR10 (right) architectures with and without BatchNorm. We again include error bars indicating the standard deviation across the 4 random initialization seeds. Figure 3 reveals a pronounced difference in layer sparsity between the standard CNNs and VGG-16 models. For the standard CNNs, the representations of the BatchNorm models are generally less sparse than those of the models trained without BatchNorm, with the exception of the final layer. Conversely, for the VGG-16 models, the BatchNorm models exhibit greater overall sparsity compared to the Non-BatchNorm models. The sparsity values are consistent across different seeds for both architectures, except for the last layers of the VGG-16 models where the variation is higher between seeds. Notably, the sparsity in the VGG-16 Non-BatchNorm models decreases significantly in the final layers, while the sparsity in the BatchNorm models remains consistent and relatively high.

It has been speculated that sparser representations are more desirable, as they suggest that the network is selectively activating in response to specific features. However, it is evident that BatchNorm neither consistently induces nor prevents sparsity in the representations. Yet, in both cases, the models with BatchNorm generalize better. This points to the notion that the presence or absence of sparse representations does not appear to be a primary factor that determines the generalization ability of a network. Instead, it is likely that other effects of BatchNorm play a more significant role in the network's generalization performance.



(a) Standard CNN trained on MNIST



(b) VGG-16 trained on CIFAR10

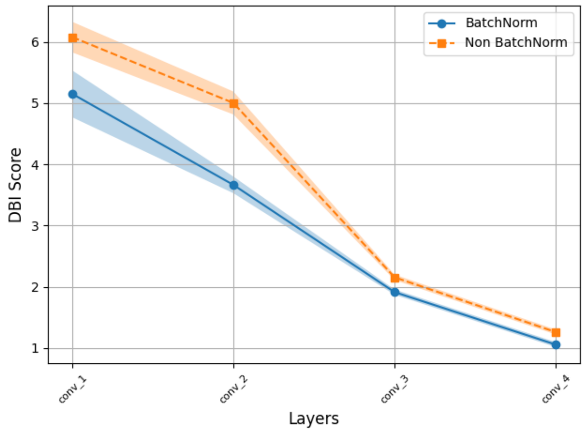
Figure 3: Average layer sparsity for models trained with (blue) and without (orange) BatchNorm. Error bars indicate the standard deviation across 4 random initialization seeds.

5 Clustering Analysis and Results

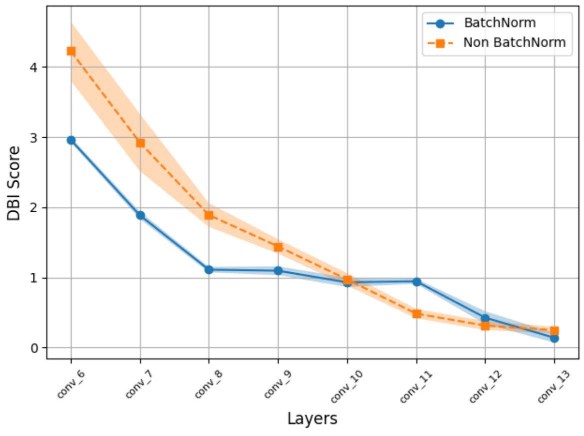
In this section, we analyse the effect of BatchNorm on the purity of representational clusters formed by the network. We consider this from both a class-specific (Section 5.1) and class-agnostic (Section 5.2) perspective.

5.1 Class-based Analysis

We first analyze the purity of the clusters formed in each layer of the network, under the assumption that the class of each sample corresponds to the cluster to which it belongs. We use 15 000 randomly selected training samples and preprocess the representations as described in Section 3.3. Figure 4 presents the DBI scores for each layer of the networks trained with and without BatchNorm, for both the MNIST and CIFAR10 architectures, respectively. The average DBI scores across the 4 different initialization seeds, along with the corresponding standard deviation, are displayed. For the VGG-16 network, we only show from convolutional layer six onward as the earlier layers are not informative and show unstable behavior among different seeds.



(a) Standard CNN trained on MNIST



(b) VGG-16 trained on CIFAR10

Figure 4: Average class-based DBI cluster purity score per layer for BatchNorm (blue) and Non-BatchNorm (orange) models, with error bars showing standard deviation across 4 random initialization seeds. Lower scores indicate higher purity.

For both network architectures, the models trained with BatchNorm generally exhibit higher cluster purity compared to those trained without. An exception is observed in layers 11 and 12 of the VGG-16 model; however, the overall cluster purity remains poorer in the BatchNorm models. Furthermore, we also observe that the purity values are relatively consistent across the different seeds, with minimal variation. Since only the 10 dataset classes serve as the cluster labels, and each sample can only belong to the cluster corresponding to its class label, both models start off with very low purity when traversing through the layers. However, in the deeper layers, clusters become increasingly pure, ultimately nearing complete purity, as observed in layer 13 of the VGG-16 models.

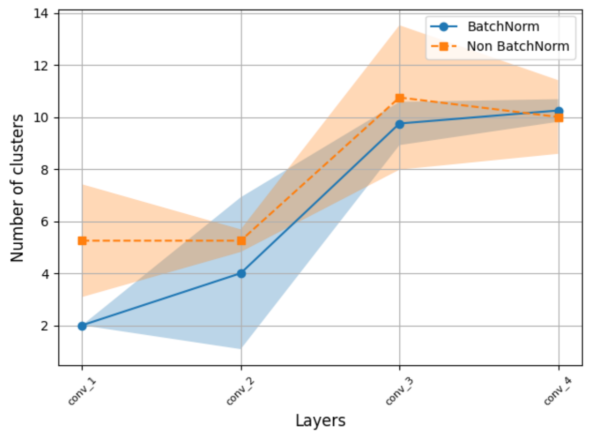
We suspect that the common features shared among different classes could lead to the formation of either fewer or more clusters than there are classes in the earlier layers. Overall, observations are consistent with the well-known belief that earlier layers learn more general patterns in CNNs [26]. In deeper layers, we suspect that more class-specific features, with fewer shared features between classes, likely contribute to the higher clustering purity observed.

5.2 Class-agnostic Analysis

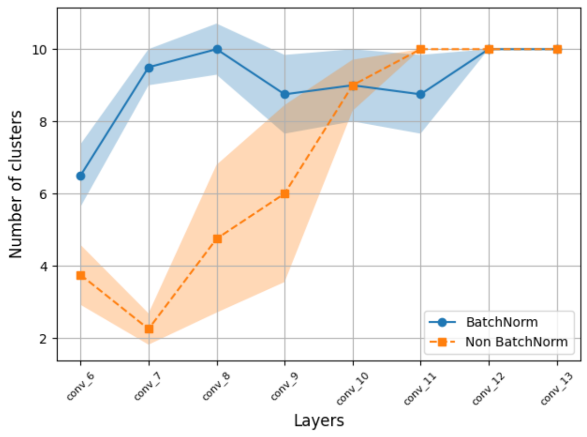
We now turn our attention to the class-agnostic clustering perspective. As described in Section 3.3, we assign clusters using the k-means clustering algorithm, and then measure the resulting purity. As with the previous experiment, a total of 15 000 randomly selected training samples are utilized for these calculations.

5.2.1 Optimal number of clusters

The optimal number of clusters for each layer is shown in Figure 5 for both the MNIST (left) and CIFAR10 (right) architectures. The error bars again report the standard deviation across the 4 random initialization seeds. We only show from convolutional layer 6 onward for the VGG-16 models as the initial layers are not informative and unstable between seeds. In Figure 5 we observe that, for both architectures, the optimal number of clusters initially starts small



(a) MNIST trained standard CNN



(b) CIFAR10 trained VGG-16

Figure 5: Average optimal number of class-agnostic clusters per layer for BatchNorm (blue) and Non-BatchNorm (orange) models, with error bars showing the standard deviation across 4 random initialization seeds.

but then increases. At the final layers, the optimal number converges to 10 clusters, corresponding to the number of classes in the dataset. For the standard CNN models, the increase in the optimal number of clusters occurs in the middle of the network; for the VGG-16 models, this occurs closer to the end. This is true for both the models trained with and without BatchNorm.

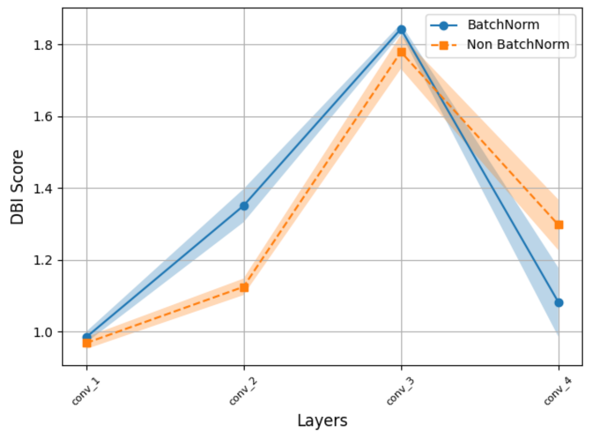
For VGG-16 models, the optimal cluster patterns differ across model types In BatchNorm models, the number of optimal clusters increases in the early layers, stabilizing until the last two layers, where it matches the number of classes. For Non-BatchNorm models, the optimal number of clusters is initially reached but significantly decreases in the middle layers, then increases again in the later layers, ultimately matching the number of classes in the final layers.

These observations support the hypothesis presented in Section 5.1 that optimal class-wise clusters are achieved only in the later layers. It suggests that while different class features contribute to cluster formation, the clusters do not become class-specific until the final layers of the network. These plots also suggest that the improved generalization ability of

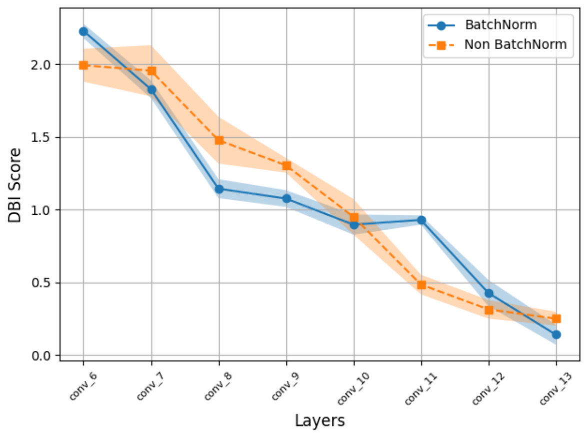
BatchNorm models is likely due to the greater consistency of clusters formed by the representations throughout the network, compared to those in models trained without BatchNorm.

5.2.2 Cluster purity

We now consider the DBI score of each optimal cluster found using k-means. This is shown for the MNIST and CIFAR10 models in Figure 6. We again only show from convolutional layer 6 onward for the VGG-16 models as the initial layers are not informative and unstable between seeds. In Figure 6 we observe that the standard CNNs, for



(a) MNIST trained standard CNN



(b) CIFAR10 trained VGG-16

Figure 6: Average class-agnostic DBI cluster purity score for k-means per layer for BatchNorm (blue) and Non-BatchNorm (orange) models, with error bars indicating standard deviation across 4 random initialization seeds. Lower scores indicate higher purity.

both model types, follow the same pattern where the DBI score increases significantly until layer 3 and then decreases. The BatchNorm models also have a slightly higher score than the Non-BatchNorm except for the last layer where the BatchNorm has a lower score. Comparing these results with what was seen in Figure 4 with Figure 6 there is a difference: the scores in Figure 4 only decreases whereas the DBI scores in Figure 6 increases before decreasing in the last layer. One thing to note is that the DBI scores in Figure 4 are significantly higher (poorer) than those in Figure 6.

For the VGG-16 network, the overall DBI score of the BatchNorm models is better compared to the non-BatchNorm models. Variation between seeds is more stable for the BatchNorm models than non-BatchNorm models. This suggests that the clusters are more stable for the BatchNorm models over seeds. Comparing Figure 6 with Figure 4, the same pattern emerges for the VGG-16 models where the BatchNorm models have purer clusters except for layers 11 and 12. This suggests better cluster forming in general, and thus better generalization in the BatchNorm models, especially in the last layer.

6 Discussion

Let us summarise the main observations. In the context of convolutional neural networks and the experiments conducted here:

1. BatchNorm does not consistently induce nor prevent sparsity in hidden representations.
2. Sparse representations do not correlate with better generalization.
3. BatchNorm tends to produce purer representational clusters than models trained without, especially in the case of class-based representations. For class-agnostic representation, results are less clear, except at the final layer, where BatchNorm clusters are again purer.
4. BatchNorm forms class-agnostic clusters earlier in the network and these early clusters are more consistent with the actual number of classes.

It is challenging to determine the exact cause of the sparsity in the representations studied because we observed significant differences between models trained on MNIST and those trained on CIFAR10. However, a clear distinction exists between models with BatchNorm and those without. For standard CNNs, applying BatchNorm results in much less sparse representations, which aligns with the findings of Pretorius et al. [15]. In contrast, VGG-16 models exhibit the opposite behaviour, where BatchNorm models produce significantly more sparse representations than their non-BatchNorm counterparts.

The difference in sparsity might be influenced by the dataset on which the models are trained, or to the architectural differences between the models. Specifically, VGG-16 models with BatchNorm seem to activate for more specific features, leading to increased sparsity, particularly around layer 5. On the other hand, non-BatchNorm models in VGG-16 also show increased sparsity until layer 5 but revert to activating more general features in deeper layers, causing a reduction in sparsity. However, note the variability in the results across seeds for the deeper layers – this could be indicative of other underlying processes at play.

Generally, BatchNorm models tend to produce purer clusters compared to non-BatchNorm models. In the VGG-16 models, both clustering methods yielded similar results with BatchNorm performing better overall, with the exception of layers 11 and 12. For the layers where non-BatchNorm perform better in class-agnostic clustering, it results in more clusters and a wider range of cluster numbers across seeds compared to the BatchNorm models.

The work also raises additional questions: Do clearer patterns emerge if the experiment is repeated on additional datasets and architectures? Since the representations are highly dimensional, clustering them is computationally expensive, and repeating these experiments over multiple seeds quickly becomes expensive, it remains an avenue we wish to pursue further. Similarly, we used 0 as the threshold for measuring sparsity, even though ‘very small’ numbers could also be considered sparse. In the current analysis, we avoided re-running experiments across multiple thresholds, although it could be worth exploring in more detail.

7 Conclusion

Studying the characteristics of DNNs’ internal representations can provide insight into the underlying mechanisms that contribute to good generalization. In this study, representations were analysed from two convolutional architectures: a small fairly standard CNN architecture and a VGG-16 architecture. We developed models with similar training accuracy but – since the models with BatchNorm generalized better on unseen data – the models represented different generalization gaps. The main question asked was: how does BatchNorm affect the characteristics of the internal representations of these two sets of models?

Representational sparsity did not correlate either with the generalization ability of the networks or could be linked to whether BatchNorm was applied or not. Clearer trends emerged with regard to the consistency of representations, specifically when considering the purity of clusters formed by these representations at different layers in the network. Overall, BatchNorm tended to produce purer representational clusters, both in the case of class-based and class-agnostic analysis. In addition, class-specific clusters formed earlier in the BatchNorm networks, and these were more consistent with the actual number of classes.

Trends observed were complex and pose additional questions that can be explored further. In this work, our objective was to provide initial results and motivate the importance and potential of studying the characteristics of DNN representations to shed light on the underlying mechanisms controlling the generalization ability of these models.

Acknowledgements

This work is based on the research supported in part by the National Research Foundation of South Africa (Ref Numbers PSTD23042296065, RA211019646111).

References

[1] Hermanus L Potgieter, Coenraad Mouton, and Marelie H Davel. Impact of batch normalization on convolutional network representations. In *Southern African Conference for Artificial Intelligence Research SACAIR*, pages 235–252. Springer, 2024.

[2] Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning (still) requires rethinking generalization. *Communications of the ACM*, 64(3), 2021.

[3] Pierre Foret, Ariel Kleiner, Hossein Mobahi, and Behnam Neyshabur. Sharpness-aware minimization for efficiently improving generalization. In *9th International Conference on Learning Representations, ICLR*, 2021.

[4] Olivier Bousquet and André Elisseeff. Algorithmic stability and generalization performance. *Advances in Neural Information Processing Systems NeurIPS*, 13, 2000.

[5] Coenraad Mouton, Marthinus Wilhelmus Theunissen, and Marelie H Davel. Input margins can predict generalization too. *Proceedings of the AAAI Conference on Artificial Intelligence*, 38(13), 2024.

[6] Kenji Kawaguchi, Zhun Deng, Xu Ji, and Jiaoyang Huang. How does information bottleneck help deep learning? In *Proceedings of the 40th International Conference on Machine Learning ICML*, volume 202, 2023.

[7] Ching-Yao Chuang, Youssef Mroueh, Kristjan Greenewald, Antonio Torralba, and Stefanie Jegelka. Measuring generalization with optimal transport. In *Advances in Neural Information Processing Systems NeurIPS*, volume 34, 2021.

[8] Yiding Jiang, Dilip Krishnan, Hossein Mobahi, and Samy Bengio. Predicting the generalization gap in deep networks with margin distributions. In *7th International Conference on Learning Representations, ICLR*, 2018.

[9] Parth Natekar and Manik Sharma. Representation based complexity measures for predicting generalization in deep learning. *arXiv preprint arXiv:2012.02775*, 2020.

[10] X. Glorot, A. Bordes, and Y. Bengio. Deep sparse rectifier neural networks. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, volume 15 of *Proceedings of Machine Learning Research*, 2011.

[11] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32nd International Conference on Machine Learning ICML*, volume 37, 2015.

[12] Zhiyuan Li and Sanjeev Arora. An exponential learning rate schedule for deep learning. In *8th International Conference on Learning Representations, ICLR*, 2019.

[13] Shibani Santurkar, Dimitris Tsipras, Andrew Ilyas, and Aleksander Madry. How does batch normalization help optimization? *Advances in Neural Information Processing Systems NeurIPS*, 31, 2018.

[14] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT Press, 2016.

[15] Arnold M Pretorius, Etienne Barnard, and Marelie H Davel. Relu and sigmoidal activation functions. In *Proceedings of South African Forum for Artificial Intelligence Research FAIR*, 2019.

[16] Renjie Liao, Alex Schwing, Richard Zemel, and Raquel Urtasun. Learning deep parsimonious representations. *Advances in Neural Information Processing Systems NeurIPS*, 29, 2016.

[17] Nils Bjorck, Carla P Gomes, Bart Selman, and Kilian Q Weinberger. Understanding batch normalization. In *Advances in Neural Information Processing Systems NeurIPS*, volume 31, 2018.

[18] Philipp Benz, Chaoning Zhang, and In So Kweon. Batch normalization increases adversarial vulnerability and decreases adversarial transferability: A non-robust feature perspective. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021.

[19] Randall Balestriero and Richard G Baraniuk. Batch normalization explained. *arXiv preprint arXiv:2209.14778*, 2022.

[20] Hadi Daneshmand, Amir Joudaki, and Francis Bach. Batch normalization orthogonalizes representations in deep random networks. *Advances in Neural Information Processing Systems NeurIPS*, 34, 2021.

[21] Y. Bengio, Aaron Courville, and Pascal Vincent. Representation learning: A review and new perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(8), 2013.

[22] James MacQueen et al. Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley Symposium on Mathematical Statistics and Probability*, volume 1, 1967.

[23] Frank Nielsen. *Hierarchical clustering*, pages 195–211. 2016.

[24] Jianbo Shi and Jitendra Malik. Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(8), 2000.

[25] Coenraad Mouton and Marelie Davel. Exploring layerwise decision making in dnns. *Communications in Computer and Information Science CCIS*, 1551, 2022.

[26] Mathilde Caron, Piotr Bojanowski, Armand Joulin, and Matthijs Douze. Deep clustering for unsupervised learning of visual features. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018.

[27] David Arthur and Sergei Vassilvitskii. K-means++: the advantages of careful seeding. In *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms*, volume 8, 2007.

[28] Joseph C Dunn. A fuzzy relative of the ISODATA process and its use in detecting compact well-separated clusters. *Journal of Cybernetics*, 3(3), 1973.

[29] Anil K Jain and Richard C Dubes. *Algorithms for clustering data*. Prentice-Hall, Inc., 1988.

[30] David L Davies and Donald W Bouldin. A cluster separation measure. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-1(2), 1979.

[31] Xiaohang Zhan, Jiahao Xie, Ziwei Liu, Yew-Soon Ong, and Chen Change Loy. Online deep clustering for unsupervised representation learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020.

[32] Junnan Li, Pan Zhou, Caiming Xiong, and Steven Hoi. Prototypical contrastive learning of unsupervised representations. In *9th International Conference on Learning Representations, ICLR*, 2021.

[33] Yann LeCun, Corinna Cortes, and Christopher Burges. The MNIST database of handwritten digits. *http://yann. lecun. com/exdb/mnist/*, 2, 1998.

[34] Alex Krizhevsky. Learning multiple layers of features from tiny images. 05 2012.

[35] Preetum Nakkiran, Gal Kaplun, Yamini Bansal, Tristan Yang, Boaz Barak, and Ilya Sutskever. Deep double descent: where bigger models and more data hurt. *Journal of Statistical Mechanics: Theory and Experiment*, 2021, 2019.

[36] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *3rd International Conference on Learning Representations, ICLR*, 2015.