Homogeneous Vector Capsules Enable Adaptive Gradient Descent in Convolutional Neural Networks

ADAM BYERLY[1,2] **AND TATIANA KALGANOVA**[1]

[1]Department of Electronic and Electrical Engineering, Brunel University London, UB8 3PH Uxbridge, U.K.
[2]Department of Computer Science and Information Systems, Bradley University, Peoria, IL 61615, USA

Corresponding author: Adam Byerly (abyerly@fsmail.bradley.edu)

ABSTRACT Neural networks traditionally produce a scalar value for an activated neuron. Capsules, on the other hand, produce a vector of values, which has been shown to correspond to a single, composite feature wherein the values of the components of the vectors indicate properties of the feature such as transformation or contrast. We present a new way of parameterizing and training capsules that we refer to as homogeneous vector capsules (HVCs). We demonstrate, experimentally, that altering a convolutional neural network (CNN) to use HVCs can achieve superior classification accuracy without increasing the number of parameters or operations in its architecture as compared to a CNN using a single final fully connected layer. Additionally, the introduction of HVCs enables the use of adaptive gradient descent, reducing the dependence a model's achievable accuracy has on the finely tuned hyperparameters of a non-adaptive optimizer. We demonstrate our method and results using two neural network architectures. For the CNN architecture referred to as Inception v3, replacing the fully connected layers with HVCs increased the test accuracy by an average of 1.32% across all experiments conducted. For a simple monolithic CNN, we show HVCs improve test accuracy by an average of 19.16%.

INDEX TERMS Adaptive gradient descent, capsule, convolutional neural network (CNN), homogeneous vector capsules (HVCs), Inception.

I. INTRODUCTION

In [1], the authors argued that standard convolutional neural networks are "misguided" in their usage of neurons that are composed of singular scalars to summarize their activation. The authors proposed (a) the concept of a "capsule", which is comprised of multiple scalar values and (b) posited that these capsules would be capable of recognizing a "visual entity over a limited domain of viewing conditions and deformations" and that the capsule's members would include both the probability that the entity is present as well as a set of "instantiation parameters" that "may include the precise pose, lighting and deformation relative to the canonical version of that entity". In their work, they (c) demonstrated that capsules could learn the x and y coordinates of a visual entity and (d) made a convincing case that capsules could learn to identify "any property of an image that we can manipulate in a known way".

Research into capsules did not progress much until a pair of papers were pre-published on arXiv in late 2017. The first of these two papers ([2]) received an especially significant

The associate editor coordinating the review of this manuscript and approving it for publication was Wenming Cao.

amount of attention, due to the fact that it published results on par with the state-of-the-art for both the standard MNIST [3] and smallNORB [4] datasets using a relatively shallow network in combination with capsules. Additionally, the network described in the first paper was shown to be highly effective at segmenting highly overlapped digits from the MNIST data. Both papers utilized an iterative routing mechanism between layers of capsules. They referred to the method in the first paper as "Dynamic Routing" and used a different method in the second paper based on the Expectation-Maximization algorithm [5]. The architecture described in the second paper ([6]) improved upon the state-of-the-art classification accuracy for smallNORB by 45%.

The architectures described in both papers used two layers of capsules in order to make the final classification and used matrix multiplication between them. In both papers, in addition to learning the weights used in the matrix multiplications using backpropagation, a routing algorithm was employed to iteratively "refine" the weights of the matrices. The authors interpret the first set of capsules as "parts" and the second set as "wholes" and the routing algorithm as a method for finding agreement about which whole is best described by the particular set of parts.

Both papers published results on relatively small data sets. In both cases this was due to the high computational cost associated with using a routing algorithm. Additionally, the architecture from the first paper requires a large number of parameters per output class (147,456) just for the weights between capsule layers, making datasets with a large number of output classes (like the 1,000 classes in ImageNet) intractable.

Another important thread of neural network research is choosing the best optimization algorithm and its hyperparameters. Stochastic Gradient Descent (SGD) with momentum is simple and effective but requires careful tuning of both the learning rate η and the schedule for decaying that learning rate as training progresses. Though guidance has emerged in the form of rules-of-thumb [7], it is none-the-less true that the choice of the learning rate and rate decay scheme remain a matter of trial-and-error and heavily dependent on the data being trained on. As such, alleviating the need to carefully tune a single learning rate has emerged as an important research area.

The most successful strategy for alleviating the need to carefully tune the learning rate has been to maintain separate learning rates for every trainable parameter and to learn each of these learning rates based on the magnitude of previous gradient updates to those parameters. This method in general is referred to as adaptive gradient descent. Research into this began in earnest with AdaGrad [8] and has continued to be an active area of research up to the present, with the most popular adaptive method currently being Adam [9]. Adaptive methods of gradient descent are popular for several reasons. First, because they adapt a learning rate for every parameter, they are able to learn sparse, yet highly informative features differently than more dense information that may be less predictive. Second, they reduce the need for careful tuning of the learning rate and learning rate decay by allowing the learning rate to be ''learned'' from the data. And third, they tend to approach a convergence much earlier in the training scheme compared to non-adaptive methods for the same data and network.

Unfortunately, adaptive gradient descent methods have some weaknesses. First, sparsely occurring features that are not highly informative have overweight influence relative to less sparsely occurring features. And second, empirically, they are prone to overfitting and creating a generalization gap between the in-sample and out-of-sample predictions. This has led some researchers to state that the generalization gap of adaptive gradient descent methods is an open problem [10] and has led other researchers to recommend not using adaptive methods at all [7]. Indeed, the best performing convolutional neural networks (CNNs) of the past few years have all used non-adaptive gradient descent methods and hand-tuned learning rate decay schemes [11], [12] [13], [14] [15], [16].

Our contribution is as follows:

1) We present a new way of parameterizing and training a pair of capsule layers which we call homogeneous vector capsules (HVCs). This method, as compared to other prevailing capsule methods (see [2], [6], [17], and [18]), uses drastically fewer parameters and avoids expensive iterative routing procedures, instead relying solely on weights learned during backpropagation.

2) We demonstrate experimentally that classifying with HVCs, rather than classifying with fully connected layers, achieves massively superior results in a simple monolithic CNN and quantifiably superior results in a more advanced CNN architecture (Inception v3).

3) We show that the practice of using large values of ϵ in popular adaptive gradient descent methods has the effect of muting their adaptability.

4) We demonstrate experimentally that when using HVCs, training with adaptive gradient descent methods using the intended small value of ϵ restores the adaptability of the methods and achieves superior classification accuracy relative to finely-tuned learning rates and decay schedules. In so doing, we solve an open problem in convolutional neural network research.

II. RELATED WORK

Morzhakov and Redozubov [19], inspired by the work of Hubel & Wiesel [20], put forth a neural network architecture similar to that used by [2] in that it utilized vector neurons, rather than scalar neurons, which shared common inputs and outputs. As their work was inspired by the physiology of primate brains, they characterized the structure as minicolumns, the term used for the analogous structure in primate brains. It is noteworthy that their architecture did not use any analog to the routing mechanism employed by [2] and [6]. While performing comparably with traditional CNNs on the MNIST dataset, it performed worse than the architecture employed by [2].

Roy *et al.* [21], compared the effects of various forms of image degradation (additive white Gaussian noise, salt and pepper noise, etc.) on MobileNet [22], VGG16 & VGG19 [11], Inception v3 [13], and CapsNet [2] and found that CapsNet was far more robust against the degradation methods they tested than any of the others. They hypothesize that this is not only due to the presence of the capsule neurons and/or dynamic routing, but also due to the shallower nature of CapsNet, having gone through fewer layers of convolutions.

In addition to MNIST and smallNORB, researchers ([23], [24], [25], [26]) have ventured to apply capsules to marginally more complex datasets such as Fashion MNIST [27], SVHN [28], CIFAR-10, and CIFAR-100 [29]. smallNORB notwithstanding, these datasets consist of images that are no more than 32×32 pixels, with smallNORB having images of resolution 96×96. In our experiments, we chose datasets ([30] and [31]) with images of much larger dimensions that we *scaled down* to 299×299 before feeding into our networks.

In their experimentation, Nair *et al.* [23], concluded that the CapsNet architecture is ''unlikely to work on other classification tasks, let alone machine learning tasks in general''. They also concluded that the design was ''not making full use

of routing to encode" the spatial relationships between the components of the objects the network was classifying. They hypothesized that a neural network, as opposed to a routing algorithm, would better accomplish the goal of reweighting the coefficients used to determine the agreement between capsule layers. This method was experimented with by [32], though they were unable to produce any significant results, whereas Hahn *et al.* [24] were able to use neural network routing to achieve comparable results to [2] but requiring far less computation.

Indeed, the routing procedure of CapsNet and subsequent capsule networks has become a topic of debate. Experimentation performed by Gu and Tresp [26] shows that the routing procedure of CapsNet contributes neither to the network's ability to generalize nor to its robustness to novel affine deformations. Paik *et al.* [25] analyzed five different routing algorithms (including [2] and [6]) and concluded that routing procedures can produce results that are worse than assigning connection strengths uniformly or randomly and that in most cases the routing procedures do not change the classification result but rather simply polarize the strengths of the connections between the capsules. With HVCs, we use no special routing procedure, and instead rely solely on weights between capsule layers that are learned during back-propagation.

One of the best performing CNNs to be published in the past several years is Inception v3 [13], which achieves near state of the art accuracy on the full ILSVRC 2012 ImageNet relative to the modest number of parameters it uses ($\sim$24.5M). To train their architecture, they used the RMSProp[1] optimizer, which is indeed designed to be an adaptive gradient descent method. RMSProp adapts each parameter in the model using Equation 1:

$$\frac{1}{\sqrt{E[g^2] + \epsilon}} \tag{1}$$

$E[g^2]$ is the exponential moving average of the past squared gradients for the parameter and the intended purpose of the ϵ parameter is to provide numeric stability by mitigating the danger of division by zero, and thus implementations default this value to 1×10^{-10}, which would create a range of possible values for the per-parameter adaptive term of 0 to 1×10^6. By using a value of 1.0 when training Inception v3, they limit this range to 0 to 1, thus setting an upper bound five orders of magnitude less than intended for this term. While still technically adapting each parameter, the range of adaptation is so dampened that we would characterize RMSProp with a 1.0 ϵ as quasi-adaptive at best. As such, we agree with Chen and Gu [10] that effectively utilizing (truly) adaptive gradient descent methods with convolutional neural networks remains an open problem relative to Inception v3.

The Adam optimizer [9] has an analogous per-parameter adaptive term for each of the past squared gradients shown in

Equation 2 (in addition to another term not relevant to this discussion for past gradients that gives Adam a momentum-like behavior):

$$\frac{1}{\sqrt{\hat{v}_t} + \epsilon} \tag{2}$$

$\hat{v}_t$ is the bias corrected exponential moving average of the past squared gradients for the parameter. Here again, the Adam optimizer employs the use of an ϵ that implementations default to (1×10^{-10}. Since in Adam, the ϵ is moved out from underneath the radical, Adam is able to adapt each parameter by five orders of magnitude more than RMSProp (with a range of 0 to 1×10^{10}.

III. CAPSULE LAYERS CONFIGURATION

Sabour *et al.* [2] proposed two final layers of capsules. The first of which has 8 dimensions shaped as a vector and the second of which has 16 dimensions, also shaped as a vector. The transformation between the two layers of capsules is a typical matrix multiplication, wherein every pair of capsules has an associated 16×8 matrix of trainable parameters and is multiplied by each of the 8-dimensional vector capsules and summed to form the input into the 16-dimensional capsule. In Equation 3, an equivalent transformation simplified to two and four dimensions for clarity is presented.

$$\begin{bmatrix} a & b \\ c & d \\ e & f \\ g & h \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} ax_1 + bx_2 \\ cx_1 + dx_2 \\ ex_1 + fx_2 \\ gx_1 + hx_2 \end{bmatrix} \tag{3}$$

In this equation, as well as in Equation 4 and Equation 5, the variables a through h represent learned weights that are being applied in the transformation, and the variables x_1 through x_2 (in Equation 3) and x_4 (in Equation 4 and Equation 5) represent the values computed from the previous operation in the network. In each case, the second matrix on the left hand side of the equation is the first capsule and the matrix on the right hand side is the second capsule.

A problem with this transformation in Equation 3 becomes apparent when viewing it as an overdetermined system of linear equations in matrix form: every dimension in the second layer of capsules, beyond the dimensions in the first layer, are at best redundant and more probably, due to the random initialization of the weights, a challenge to the optimization algorithm used during backpropagation to reconcile multiple differing losses derived from each activation in the previous layer.

Also, it should be noted that each dimension of the second layer of capsules is a linear combination of all dimensions of the first layer of capsules. This is a desirable property in a fully connected layer in a neural network. However, with the interpretation and empirical verification in the work of Sabour *et al.* [2] of the dimensions of a capsule as being distinct features of a given sample, it is our hypothesis that this entangling of distinct features from one layer into all features in the next layer is an undesirable property.

---

[1]RMSProp is an unpublished, adaptive learning rate method introduced by Geoffrey Hinton in Lecture 6e of a now no longer available Coursera course. See: http://www.cs.toronto.edu/~ tijmen/csc321/slides/lecture_slides _lec6pdf

In their follow-up work, Hinton *et al.* [6] switched to using an equivalent number of dimensions in neighboring capsule layers, though they did not cite their motivation for doing so as to alleviate the problem of an overdetermined system. Additionally, they shaped their capsules as matrices rather than vectors. The authors noted that this reshaping had the effect of reducing the number of trainable parameters (for every pair of capsules) from being the product of the dimensions of the two layers of capsules to being only the number of dimensions of a single layer of capsules. This method of matrix capsules requires that the number of dimensions in neighboring layers be both equivalent and a perfect square. In Equation 4, an equivalent transformation simplified to four dimensions is presented:

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \cdot \begin{bmatrix} x_1 & x_2 \\ x_3 & x_4 \end{bmatrix} = \begin{bmatrix} ax_1 + bx_3 & ax_2 + bx_4 \\ cx_1 + dx_3 & cx_2 + dx_4 \end{bmatrix} \quad (4)$$

In addition to alleviating the problem of an overdetermined system and significantly reducing the number of trainable parameters, this formulation results in only the square root of the total number of features in the first layer being entangled with each feature in the second layer.

We propose a new method for the transformation from one layer of capsules to the next. Rather than using the typical transformation matrix, the proposed method involves using a transformation vector and rather than using the typical matrix multiplication, the proposed method involves using the Hadamard product (element-wise multiplication). This method is shown in Equation 5, simplified to four dimensions for clarity:

$$\begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} \odot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} ax_1 \\ bx_2 \\ cx_3 \\ dx_4 \end{bmatrix} \quad (5)$$

This method goes back to using vectors for the shape of the capsules and requires that the neighboring layers of capsules be of equivalent dimension, thus we call these *Homogeneous Vector Capsules*. With the constraint of requiring equivalent dimensions in the capsule layers, this method comes with the following benefits:

1) Because this method uses the Hadamard product rather than typical matrix multiplication, the drawback of using the more intuitive vector shape for a capsule is removed, as the number of trainable parameters per pair of capsules stays equal to the number of dimensions in the capsules (as in Hinton *et al.* [6]), rather than being that number of dimensions squared (as in Sabour *et al.* [2]).

2) By the nature of the Hadamard product, this method cannot suffer from the problem of an overdetermined system. This is due to the fact that any valid expression of element-wise multiplication requires the same number of dimensions be present in both operands, where the one operand represents the coefficients and
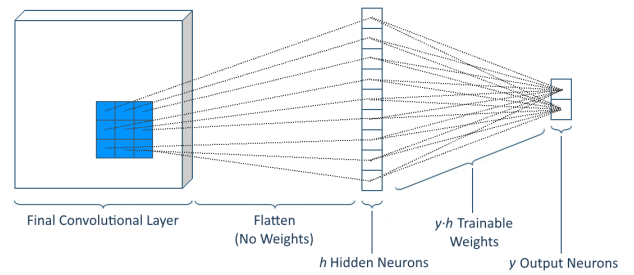

FIGURE 1. The standard approach to transforming the final convolutional layer into class predictions.
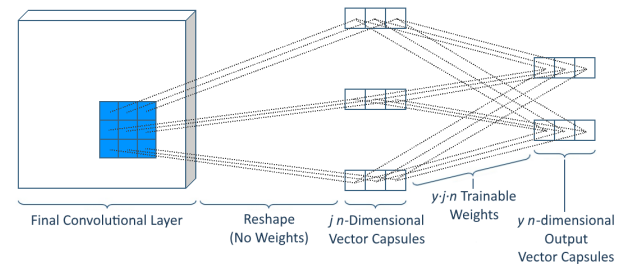

FIGURE 2. Using homogeneous vector capsules to transform the final convolutional layer into class predictions.

the other the variables (see Equation 5 contra Equation 3 which has a greater number of coefficients than variables).

3) This fully disentangles features from the dimensions in the first layer of capsules from differing dimensions in the subsequent layer of capsules.i.e., each dimension in the first layer maps to one and only one dimension in the second layer.

4) This eliminates all of the addition operations used in matrix multiplication for a modest reduction in computational cost.

5) Whereas the number of dimensions in [6] must be a perfect square, HVCs can be composed of any number of dimensions that evenly divides the number of neurons being input into them.

IV. EXPERIMENTAL SETUP AND RESULTS

We designed our experiments to compare (a) baseline neural network architectures that use the standard approach of transforming the final convolutional layer in the network as in Figure 1 with (b) reshaping the final set of feature maps into $j \cdot n$-dimensional vector capsules, where $j \cdot n$ is the total number of weights coming out of the final set of feature maps. When doing this, the final classification is done, rather than with scalar output neurons, with y n-dimensional vector capsules as in Figure 2, that are reduced to predictions by computing the Euclidian norm of the vectors.

We conducted our experiments using two convolutional neural network architectures. The first network is a typical simple monolithic CNN featuring a series of 3×3 convolutions interspersed with max pooling operations (see Table 1).

TABLE 1. The stem of the simple monolithic CNN. The baseline experiments were classified through a fully connected layer after flattening the final set of feature maps as in **Figure 1.** All other experiments used HVCs as in **Figure 2.**

Operation	Feature Maps	Output Shape
3×3 convolution w/stride 2	32	149×149×32
3×3 convolution w/stride 1	32	147×147×32
3×3 convolution w/stride 1	32	145×145×32
2×2 max pool w/stride 1	N/A	72×72×32
3×3 convolution w/stride 1	64	70×70×64
3×3 convolution w/stride 1	64	68×68×64
3×3 convolution w/stride 1	64	66×66×64
2×2 max pool w/stride 1	N/A	33×33×64
3×3 convolution w/stride	128	31×31×128
3×3 convolution w/stride 1	128	29×29×128
3×3 convolution w/stride 1	128	27×27×128
2×2 max pool w/stride 1	N/A	13×13×128
3×3 convolution w/stride 1	256	11×11×256
3×3 convolution w/stride 1	256	9×9×256

TABLE 2. Optimizers used for all experiments.

Optimizer #	Description
O1	RMSProp w/epsilon 1 and 0.045 learning rate exponentially decaying every 2 epochs by 0.94
O2	RMSProp w/epsilon 1 and 0.1 learning rate exponentially decaying every 30 epochs by 0.16
O3	Adam w/0.001 learning rate
O4	Adam w/0.001 learning rate exponentially decaying every epoch by 0.96

The motivation behind this design was to examine the effect of capsules on a simple, widely understood and easily implemented architecture with a low number of parameters (in this case ~ 1.6M to ~ 22.1M, depending on the number of output classes and the capsule configuration). We used no drop-out or L2 (or any other form) of regularization with this architecture. The second network is the popular Inception v3 architecture [13]. This network was chosen due to its good performance given the relatively low number of parameters it uses (~ 23.2M to ~ 156.1M, depending on the number of output classes and the capsule configuration).

We conducted our experiments using three datasets of increasing difficulty:
- The full-sized Imagenette [30], a subset of ImageNet consisting of 10 easily classified classes: tench, English springer, cassette player, chain saw, church, French horn, garbage truck, gas pump, golf ball, and parachute.
- The full-sized Imagewoof [30], a subset of ImageNet consisting of 10 more closely related classes, all of which are dog breeds: Australian terrier, Border terrier, Samoyed, Beagle, Shih-Tzu, English foxhound, Rhodesian ridgeback, Dingo, Golden retriever, and Old English sheepdog.
- Food-101 [31], a challenging and noisy dataset consisting of 101 classes of images retrieved from the now defunct foodspotting.com.

We conducted our experiments using four different optimization strategies. RMSProp has been a popular choice for optimizing convolutional neural networks since [13]. This strategy which we have denoted O1 (see Table 2) is the strategy used in [13] whereas the strategy we have denoted O2 is the strategy employed by the official TensorFlow

implementation of Inception v3 published on github.com[2] which results in slightly higher accuracy. In addition, we experimented with two other optimization strategies. O3 is the Adam optimizer with the defaults suggested in [9], and O4 is the Adam optimizer with a slowly decaying base learning rate.

Additional experimental parameters are as follows:
- All activations were ReLU preceded by batch normalization [33].
- Loss for the Inception v3 experiments was computed using the label-smoothing regularization method as in [13], whereas categorical cross-entropy was used for the simple monolithic CNN experiments.
- All experiments ran for 100 epochs.
- Evaluations were performed using the exponential moving average of past weights as in [34], with a decay factor of 0.999.
- A batch size of 32 was used for Imagenette and Imagewoof. A batch size of 96 was used for Food-101 for models S1-S8 and I1-I6 and a batch size of 68 for models I7 and I8 (see Table 3 and Table 4). These batch sizes were dictated by the constraints of the available hardware.
- We used an image size of 299×299 for all images in all datasets, in all cases augmented using the strategy employed by the official TensorFlow implementation of Inception v3 published on github.com.[3]

[2]https://github.com/tensorflow/models/blob/master/research/slim/train_image_classifier.py
[3]https://github.com/tensorflow/models/blob/master/research/slim/preprocessing/inception_preprocessing.py

TABLE 3. Models used for the experiments conducted using a simple monolithic CNN.

Model	Capsule Configuration	HVC Dimensions	# of HVCs
S1	No capsules — This is the baseline model		
S2	See 3a	8	2,592
S3	See 3a	16	1,296
S4	See 3a	32	648
S5	See 3a	64	324
S6	See 3a	128	162
S7	See 3b	256	81
S8	See 3c	81	256

TABLE 4. Models used for the experiments conducted using the Inception v3 architecture.

Model	Capsule Configuration	Main Out HVCs Dimensions	#	Aux Out HVCs Dimensions	#
I1	No capsules — This is the baseline model				
I2	See 3a	8	256	8	16
I3	See 3a	16	128	16	8
I4	See 3a	32	64	32	4
I5	See 3a	64	32	64	2
I6	See 3a	128	16	128	1
I7	See 3b	2,048	64	128	25
I8	See 3c	64	2,048	25	128
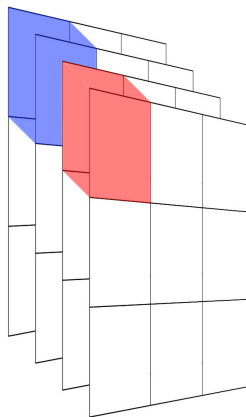
(a) In this example, the filter maps have been converted into two 2-dimensional capsules for each distinct x and y coordinate of the feature maps. The first 2 of 18 such capsules are highlighted in red and blue respectively. Models S2, I2, I3, I4, I5, and the main output of I6 use this method. The motivation behind this configuration is to allow 2 or more distinct capsules to be representative of each distinct x and y coordinate of the feature map.
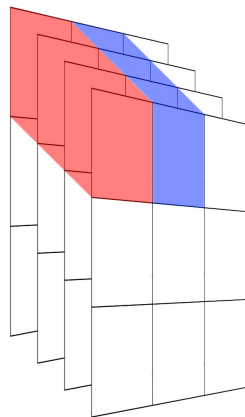
(b) In this example, the filter maps have been converted into a single 4-dimensional capsule for each distinct x and y coordinate of the feature maps. The first 2 of 9 such capsules are highlighted in red and blue respectively. Models S4, I8, and the auxiliary output of I6 use this method. The motivation behind this configuration is to force a single capsule to be representative of each distinct x and y coordinate of the feature map.
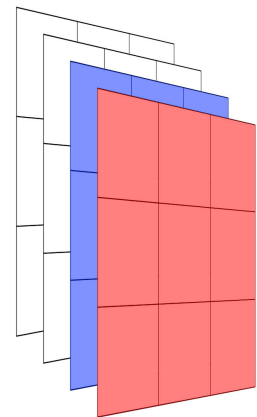
(c) In this example, the filter maps have been converted into four 9-dimensional capsules, each made from an entire feature map. The first 2 of 4 such capsules are highlighted in red and blue respectively. Models S3 and I7 use this method. The motivation behind this configuration is to force a single capsule to be representative of each distinct feature map.

FIGURE 3. Methods of converting a set of four 3 × 3 filter maps into a set of capsules.

In our experiments, we explored 3 different methods of transforming the final set of feature maps into capsules. The first method creates multiple capsules for each distinct x and y coordinate of the feature maps (see 3a). The second method creates a single capsule for each distinct x and y coordinate of the feature maps (see 3b). The intuition behind these two methods is that each position in the feature map represents a meaningful feature and that using capsules to "group" these together from multiple filter maps

encourages the feature maps to cooperate. The difference being that in the multiple capsule case, multiple disparate groups are allowed, wherein the single capsule case, only one such group is allowed. The third method creates a single capsule for each distinct feature map (see 3c). The intuition behind this method is the standard interpretation of a feature map (i.e. it represents a single feature per map). However, rather than allowing each dimension of the feature map to learn independently through a fully connected
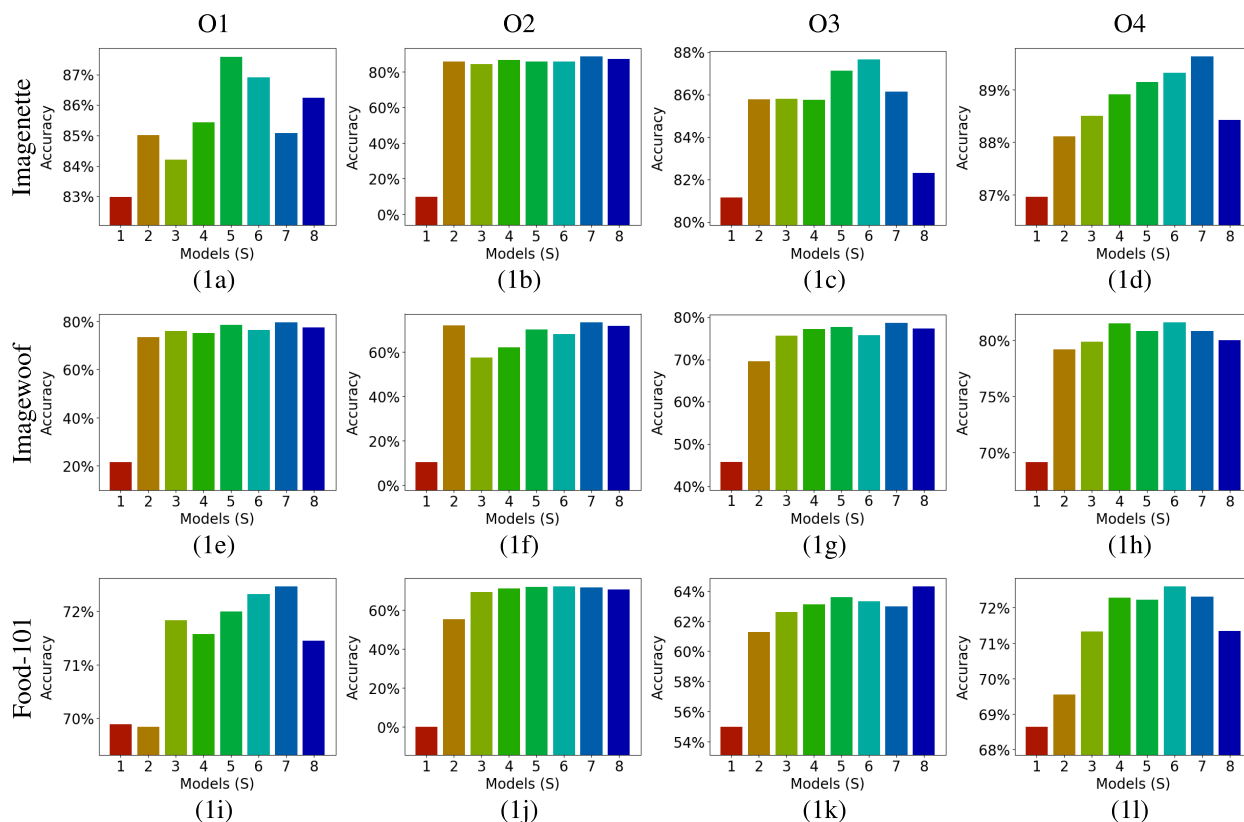
FIGURE 4. Classification accuracy on 4 simple CNN models, 3 different datasets, and 4 different optimization strategies. Each chart is cross-referenced in Table 7.

layer, we use capsules to maintain the cohesion among the dimensions.

We conducted experiments on eight variations of the simple monolithic CNN architecture (see Table 3). The first such variation, denoted S1, is the baseline model that flattens the final set of feature maps and then classifies through a layer of fully connected neurons. Variations S2 through S6 reshape the final set of feature maps as in 3a. Variation S7 reshapes the final set of feature maps as in 3b. Variation S8 reshapes the final set of feature maps as in 3c. In variations S2 through S8, after the first layer of capsules are shaped, they are then classified through the second set of capsules that form the HVC pairs.

We conducted experiments on eight variations of the Inception v3 architecture (see Table 4). The first such variation, denoted I1, is the baseline model as described in [13]. Variations I2 through I6 reshape the final set of feature maps in both the main and auxiliary branches as in 3a. Variation I7 reshapes the final set of feature maps in both branches as in 3b, and Variation I8 reshapes the final set of feature maps in both branches as in 3c. In variations I2 through I8, after the first layer of capsules are shaped each branch is then classified through the second set of capsules that form the HVC pairs.

Unique to the Inception v3 architecture relative to the simple monolithic CNN is that, in the baseline model I1 and

models I2 through I6, the final operation before the flattening operations in both the main and auxiliary outputs reduce the feature maps to 1×1. In the main branch, this is accomplished via global average pooling [35] and in the auxiliary branch, this is accomplished by performing a 5×5 convolution on a set of 5×5 feature maps. Both of these methods effectively collapse the spatial information present in the preceding operations into a single scalar value per feature map. Despite this, these global operations have been empirically shown to be effective in maintaining models' ability to achieve good generalization and accuracy, all while significantly reducing the number of trainable parameters. Generally, these global operations precede a final fully connected layer from which classification is performed. The larger the number of classes being classified, the more pronounced the reduction in trainable parameters is. For two of our Inception v3 experiments, I7 and I8, we removed these global operations, which results in the final set of feature maps in the main branch being 8×8 and the final set in the auxiliary branch being 5×5. This in turn results in an increasing number of parameters in the model as the number of output classes increases (see Table 6).

We conducted an additional experiment on the full ILSVRC 2012 ImageNet, which consists of 1.2M training images, and thus takes a considerable amount of time to train for a full 100 epochs. Using the Inception v3 models' performance on Imagenette as a guide, we chose to use the
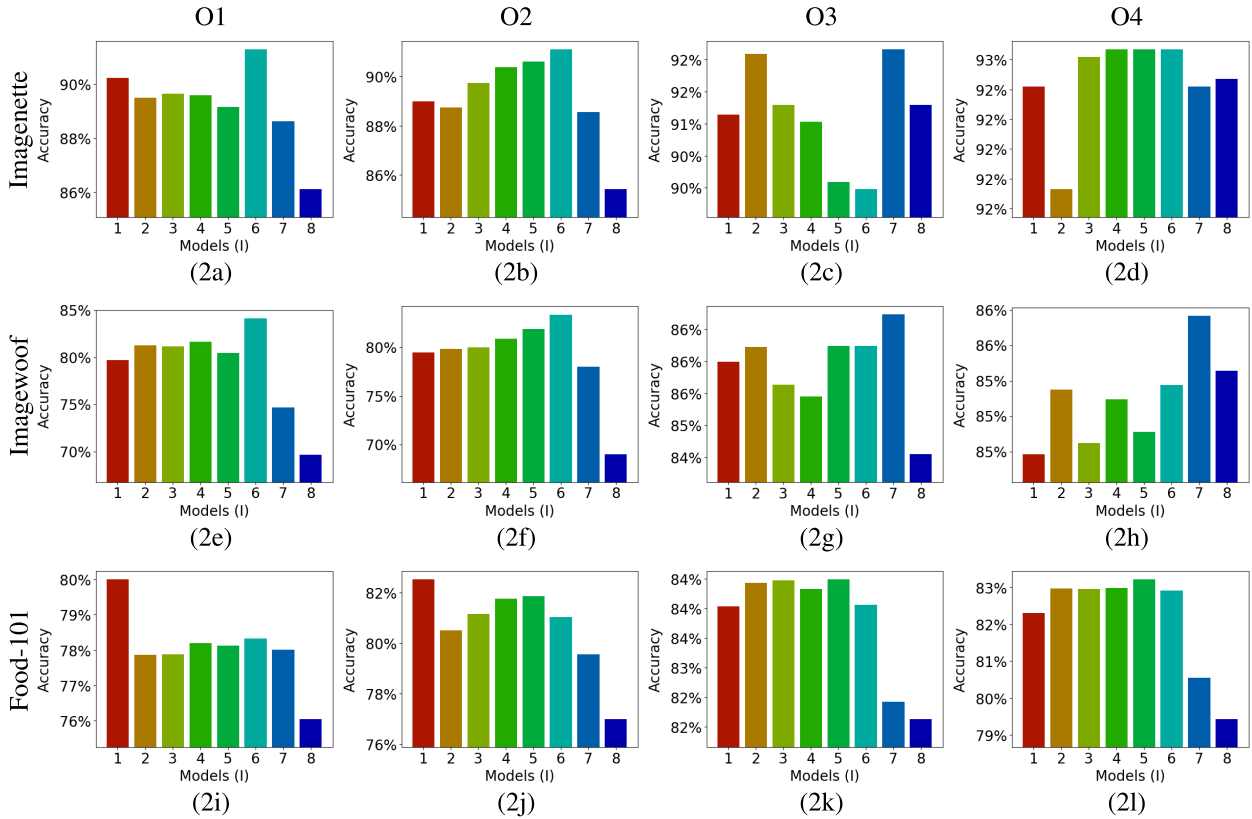
FIGURE 5. Classification accuracy on 8 inception v3 models, 3 different datasets, and 4 different optimization strategies. Each chart is cross-referenced in Table 8.

TABLE 5. Performance comparison on the full ILSVRC 2012 ImageNet.

Model	Top-5 Accuracy	Top-1 Accuracy
STAR-Caps [36]	85.66%	60.07%
Inception v3 w/HVCs	90.39%	71.3%

HVC configuration from I6 and optimizer O4. Our network significantly outperformed the only other capsule network with published full ImageNet results (see Table 5).

V. DISCUSSION

A. THE SIMPLE MONOLITHIC CNN
As can be seen in Table 7 and Figure 4, models S2-S8, which used HVCs, wildly outperformed the baseline model S1 with all optimization strategies, on all three datasets tested. The experiment for the baseline model S1 when using optimization strategy O2 was not able to learn to a better accuracy than random guessing on Imagenette and Imagewoof and actually "learned" to achieve an accuracy of 0% on Food-101. The average accuracy of the experiments of the baseline model S1 with all optimization strategies, on all three datasets, excluding those experiments where the model had not learned to an accuracy better than random guessing, is 64.55%. The average accuracy of the experiments for models S2-S8 with all optimization strategies, on all three datasets is 76.92%. This is a relative improvement of 19.16%. For all three datasets, the best performing experiments used optimization

strategy O4, which was the Adam optimizer with an appropriately small ϵ resulting in the intended adaptability along with a slowly decaying base learning rate.

6 out of 12 of the combinations of optimization strategy and dataset achieved their highest accuracy with model S7, which used the method that creates a single capsule from each distinct x and y coordinate of the feature maps. 4 out of 12 of the combinations achieved their highest accuracy with model S6, which used the method that creates 2 capsules from each distinct x and y coordinate of the feature maps. This suggests that deriving 1 or 2 capsules for each distinct x and y coordinate of the feature maps is superior to deriving a higher number of capsules from each such x and y coordinate (models S2-S5) or deriving the capsules from entire, individual feature maps (model S8).

B. INCEPTION V3
Optimization strategies O1 and O2 are the two optimization strategies published and used to train Inception v3 on ImageNet [13]. It would be understandable, yet naïve, to assume that these optimization strategies would be superior choices in general. But as can be seen in Table 8 and Figure 5, only occasionally did either strategy O1 or O2 outperform O3, and only once did O2 outperform O4. This demonstrates that finely-tuned hyperparameters are finely-tuned, not just to the network architecture, but also to the data.

TABLE 6. Trainable parameters used by the models in our experiments by number of classes. The difference between the number of trainable parameters for otherwise equivalent models using fully connected layers vs. HVCs is negligible. For S1 vs. any of S2 through S8, the former has only 0.16% fewer parameters. As models get larger the difference lessens and eventually inverts. For example, the difference between I1 and any of I2 through I6 when classifying 1000 classes is 0.007% fewer parameters for the HVC models.

Models	Final Feature Map Dimensions	Classes	# of Parameters
S1-S8 (see Table 3 and Table 7)	9×9	10 / 101 / 1000	1.6M / 3.5M / 22.1M
I1-I6 (see Table 4 and Table 8)	1×1	10 / 101 / 1000	22.3M / 22.5M / 24.5M
I7-I8 (see Table 4 and Table 8)	8×8 / 5×5	10 / 101 / 1000	23.2M / 35.4M / 156.1M

TABLE 7. Classification accuracy on 8 simple monolithic CNN models, 3 different datasets, and 4 different optimization strategies. The first column in each row is a cross-reference to the charts in Figure 4. The model parentheticals refer to the number of capsule dimensions. For example S2 (8d) refers to model S2 which uses 8-dimensional capsules.

Dataset	Figure 4 Chart	Optimizer	Models (see Table 3)							
			S1	S2 (8d)	S3 (16d)	S4 (32d)	S5 (64d)	S6 (128d)	S7 (256d)	S8 (81d)
Imagenette	1a	O1	82.99%	85.02%	84.22%	85.43%	**87.58%**	86.91%	85.09%	86.24%
	1b	O2	9.68%	85.66%	84.38%	86.68%	85.86%	85.71%	**88.65%**	87.12%
	1c	O3	81.15%	85.78%	85.81%	85.76%	87.12%	**87.65%**	86.14%	82.30%
	1d	O4	86.96%	88.11%	88.50%	88.91%	89.14%	89.32%	**89.63%**	88.42%
Imagewoof	1e	O1	21.49%	73.39%	76.00%	75.18%	78.43%	76.31%	**79.59%**	77.41%
	1f	O2	10.45%	72.03%	57.45%	62.06%	70.31%	68.21%	**73.44%**	71.85%
	1g	O3	45.72%	69.54%	75.64%	77.23%	77.72%	75.77%	**78.64%**	77.28%
	1h	O4	69.16%	79.18%	79.84%	81.48%	80.81%	**81.58%**	80.81%	79.97%
Food-101	1i	O1	69.89%	69.84%	71.83%	71.57%	71.99%	72.32%	**72.46%**	71.45%
	1j	O2	0%	55.41%	69.29%	71.23%	71.95%	**72.22%**	71.75%	70.73%
	1k	O3	54.98%	61.27%	62.60%	63.10%	63.59%	63.32%	62.98%	**64.30%**
	1l	O4	68.64%	69.55%	71.33%	72.28%	72.22%	**72.60%**	72.31%	71.33%

The only times the baseline model I1 outperformed all capsule models I2-I8 was for the Food-101 dataset when using optimization strategies O1 and O2. The best performing capsule model outperformed the baseline model I1 by an average of 1.32% across all optimization strategies and datasets.

3 out of 12 of the combinations of optimization strategy and dataset achieved their highest accuracy with model I7. This stands in contrast to the experiments on the simple monolithic CNN where twice as many combinations were superior for the analogous model S7. The two architectures are too dissimilar to draw any firm conclusions, but we hypothesize that there two factors contributing to this. First, creating a single capsule for each distinct x and y coordinate of all feature maps of the main output for Inception v3 results in 2,048 dimensional capsules (as the final set of feature maps is 2,048 in number) compared to only 256 dimensions for the capsules coming out of the final set of feature maps in the simple monolithic CNN. Second, the presence of the auxiliary output stem in Inception v3. 5 out of 12 combinations achieved their highest accuracy with model I6 and 3 out of 12 with I5.

These results are less conclusive than those with the simple monolithic CNN and permit less firm conclusions. However, these experiments do suggest that a single capsule to a small number of capsules for each distinct x and y coordinate of all feature maps is the superior choice.

C. OPTIMIZATION STRATEGY

For models S1-S8 and for all three datasets tested, optimization strategy O4 achieved the highest accuracy. The second highest accuracy was achieved with strategy O1 twice and with O2 once. For models I1-I8 and for all three datasets, optimization strategy O3 achieved the highest accuracy twice and O4 once.

With the Food-101 dataset, arguably the most difficult of the three datasets tested, baseline model S1 performed better with the quasi-adaptive optimization strategy O1 than with either of the truly adaptive strategies O3 or O4. And yet, strategy O2 achieved a top accuracy of 0% for this model. O1 and O2 are the same optimization algorithm, but parameterized differently. Further, these parameterizations were not ad-hoc, but rather parameterizations that are published along

TABLE 8. Classification accuracy on 8 Inception v3 models, 3 different datasets, and 4 different optimization strategies. The first column in each row is a cross-reference to the charts in Figure 5. The model parentheticals refer to the number of capsule dimensions. For example I2 (8d) refers to model I2 which uses 8-dimensional capsules.

Dataset	Figure 5 Chart	Optimizer	I1	I2 (8d)	I3 (16d)	I4 (32d)	I5 (64d)	I6 (128d)	I7 (2,048d/ 128d)	I8 (64d/ 25d)
						Models (see Table 4)				
Imagenette	2a	O1	90.24%	89.50%	89.65%	89.60%	89.16%	**91.29%**	88.63%	86.12%
	2b	O2	88.99%	88.73%	89.73%	90.37%	90.60%	**91.01%**	88.55%	85.43%
	2c	O3	91.14%	92.09%	91.29%	91.03%	90.09%	89.98%	**92.16%**	91.29%
	2d	O4	92.42%	91.73%	92.62%	**92.67%**	**92.67%**	**92.67%**	92.42%	92.47%
Imagewoof	2e	O1	79.71%	81.28%	81.17%	81.63%	80.48%	**84.14%**	74.67%	69.65%
	2f	O2	79.48%	79.84%	80.02%	80.89%	81.89%	**83.38%**	78.02%	69.01%
	2g	O3	85.99%	86.22%	85.63%	85.45%	86.24%	86.24%	**86.73%**	84.55%
	2h	O4	84.73%	85.19%	84.81%	85.12%	84.89%	85.22%	**85.71%**	85.32%
Food-101	2i	O1	**80.00%**	77.86%	77.88%	78.20%	78.13%	78.32%	78.01%	76.05%
	2j	O2	**82.52%**	80.51%	81.15%	81.76%	81.86%	81.03%	79.56%	76.99%
	2k	O3	84.03%	84.43%	84.47%	84.33%	**84.49%**	84.06%	82.42%	82.13%
	2l	O4	82.30%	82.97%	82.95%	82.98%	**83.22%**	82.92%	80.55%	79.43%

with the Inception v3 architecture and perform well on the ImageNet dataset with that architecture. This underscores just how important hyperparameter choice can be and how closely related to both network structure and dataset it truly is. This in turn underscores the relative utility of an adaptive gradient descent method that is less reliant on hyperparameter choice.

With adaptive gradient descent methods, there is a base learning rate η that is the same for all parameters and a separate per-parameter learning rate that is adapted based on previous gradient updates to that parameter. The two are multiplied together to determine each parameter's actual update. With the Adam optimizer, the suggested base learning rate η is 0.001 and the range of possible values for the per-parameter update are 0 to 10^{10}. After being multiplied together, this gives a range of possible per-parameter updates of 0 to 10^7. This is exactly what optimization strategy O3 uses for each parameter for the duration of the training. Optimization strategy O4 starts with this range for each parameter, and then gradually decays the base learning rate η over the epochs of training such that the resultant per-parameter updates are eventually constrained to a range of 0 to 10^4. This is similar to, but far less extreme (by four orders of magnitude) than the dampening effect caused by using a large ϵ in the denominator of the per-parameter term of an adaptive gradient descent method (contra its intended purpose), as is the case with optimization strategies O1 and O2. Further, when decaying the *base* learning rate in the manner of optimization strategy O4, the dampening is applied gradually over time as the parameter values descend the loss landscape, rather than statically for the duration of training (as in the case of a large ϵ.

Effectively, by allowing the learning rates of different parameters to change based on what has previously been learned, an adaptive gradient descent method attempts to achieve the goals of exploitation and exploration simultaneously. Exploration is achieved by decoupling each parameter

from a single learning rate and exploitation is achieved by the coupling of each parameters' own learning rate to what had previously been learned. Using an adaptive gradient descent method with a large ϵ greatly reduces the amount of per-parameter exploitation possible. This shoulders the machine learning engineer with the task of choosing just the right hyperparameters to balance this small amount of variability in exploitation with the proper amount of exploration—the very thing adaptive gradient descent methods are meant to alleviate. This is why, when using a large ϵ, we choose to characterize them as *quasi*-adaptive. By using a truly adaptive gradient descent method (one with an appropriately small ϵ and then decaying the *base* learning rate during training, the simultaneous explore/exploit nature of the method is preserved early in training and then slowly shifted to be more exploitative on average, but still allowing each parameter to have its own still rather large range of possible explore vs. exploit dispositions.

VI. CONCLUSION

The advent of convolutional layers led to considerable improvement in the performance of neural networks in image classification tasks as compared to networks composed entirely of fully connected layers [3]. This is correctly attributed to the convolutional layers' ability to extract localized features that are more complicated than a single pixel. The feature extractors do this by assigning meaning to the spatial relationships among pixels that are close to each other. Such meaning is absent when using fully connected layers. As the term "fully connected" implies, in fully connected layers every pixel is able to be associated with every other pixel without regard to their relative positions in the image. Giving meaning to spatial relationships among the pixels can be understood as enforcing constraints upon which neurons are allowed to be associated with each other using trainable

parameters. Understood in this way, the success of convolutional neural networks can thus be understood as, in part, resulting from applying constraints on which neurons are allowed to affect other neurons in the next layer.

We interpret homogeneous vector capsules as performing a similar function, at the output stage of a convolutional neural network, as convolutional layers perform at the input stage. In the traditional design of the classification stage of a CNN as depicted in Figure 1, every neuron is able to adapt independently during backpropagation. We hypothesize that this fact combined with the fact that adaptive gradient descent methods adapt independent learning rates for every parameter imparts two orders of adaptability—or stated another way, "too much" "freedom" (to adapt to the training data). This would indeed result in overfitting and a generalization gap as has been observed when using adaptive gradient descent with CNNs. By reshaping the output of the final convolutional layer into vectors and then connecting those vectors to a classification layer also composed of vectors, we are constraining groups of n-dimensional vectors of neurons to train together.

Thus, HVCs enable convolutional neural network researchers to:

1) Use adaptive gradient descent methods when training CNNs without experiencing a generalization gap.

2) Save time and compute cycles searching for the best learning rates and learning rate decay schedules to use to train their network with a non-adaptive gradient descent method and instead use an adaptive gradient descent method that does not require this fine-tuning.

In general, we hypothesize that fully connected layers of scalar valued neurons are indeed "misguided" (as per Hinton *et al.* in [1]). Specifically, that using them after the convolutional layers in a CNN works against the goal of preserving meaning in spatial relationships within the features of an image. The first layer of capsules in a pair of HVCs, groups outputs from the preceding convolutional layer together, preserving the spatial relationships that have been learned as meaningful. By "routing" them to a second layer of capsules via trainable vectors, groups of capsules (the first layer of HVCs) that have preserved feature extractions from the convolutional layers are allowed to learn when they should be associated with each other to make a classification prediction (the second layer of HVCs).

In summary, our experimentation demonstrates that:

1) Using HVCs on an advanced neural network architecture like Inception v3 increases the achievable accuracy by a small but significant margin.

2) Using HVCs on a simple monolithic CNN increases the achievable accuracy massively.

3) Deriving 1 or 2 capsules from each distinct *x* and *y* coordinate of all feature maps outperforms both deriving a larger number of capsules in the same manner and deriving capsules from entire, individual feature maps.

Future research into HVCs should include:

1) Applications to domains beyond classification.

2) The impact of 3 or more layers of HVCs.

3) Application to a greater variety of network architectures.

The code used for all experiments is publicly available on GitHub at: https://github.com/AdamByerly/HVCsEnableAGD Initially published on June 27, 2019.

REFERENCES

[1] G. E. Hinton, A. Krizhevsky, and S. D. Wang, "Transforming auto-encoders," in *Proc. Int. Conf. Artif. Neural Netw.*, 2011, pp. 44–51.

[2] S. Sabour, N. Frosst, and G. E. Hinton, "Dynamic routing between capsules," in *Proc. 31st Conf. Neural Inf. Process. Syst. (NIPS)*, 2017, pp. 1–11.

[3] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2323, Nov. 1998.

[4] Y. LeCun, F. J. Huang, and L. Bottou, "Learning methods for generic object recognition with invariance to pose and lighting," in *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2004, pp. 97–104.

[5] A. P. Dempster, N. M. Laird, and D. B. Rubin, "Maximum likelihood from incomplete data via the EM algorithm," *J. Roy. Stat. Soc., B Methodol.*, vol. 39, no. 1, pp. 1–22, 1977.

[6] G. E. Hinton, S. Sabour, and N. Frosst, "Matrix capsules with em routing," in *Proc. 6th Int. Conf. Learn. Represent. (ICLR)*, 2018, pp. 1–15.

[7] A. C. Wilson, R. Roelofs, M. Stern, N. Srebro, and B. Recht, "The marginal value of adaptive gradient methods in machine learning," in *Proc. 31st Conf. Neural Inf. Process. Syst.*, 2018, pp. 1–14.

[8] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *J. Mach. Learn. Res.*, vol. 12, no. 7, pp. 2121–2159, 2011.

[9] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *Proc. 6th Int. Conf. Learn. Represent. (ICLR)*, 2014, pp. 1–15.

[10] J. Chen and Q. Gu, "Closing the generalization gap of adaptive gradient methods in training deep neural networks," 2018, *arXiv:1806.06763*. [Online]. Available: https://arxiv.org/abs/1806.06763

[11] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *Proc. Int. Conf. Learn. Represent.*, 2015, pp. 1–14.

[12] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2015, pp. 1–9.

[13] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 2818–2826.

[14] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 770–778.

[15] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. Alemi, "Inception-v4, inception-resnet and the impact of residual connections on learning," in *Proc. AAAI*, 2016, pp. 4278–4284.

[16] F. Chollet, "Xception: Deep learning with depthwise separable convolutions," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 1800–1807.

[17] S. R. Venkataraman, S. Balasubramanian, and R. R. Sarma, "Building deep equivariant capsule networks," in *Proc. Int. Conf. Learn. Represent.*, 2020, pp. 1–12.

[18] M. Amer and T. Maul, "Path capsule networks," *Neural Process. Lett.*, vol. 52, no. 1, pp. 545–559, Aug. 2020.

[19] V. Morzhakov and A. Redozubov, "An artificial neural network architecture based on context transformations in cortical minicolumns," 2017, *arXiv:1712.05954*. [Online]. Available: https://arxiv.org/abs/1712.05954

[20] D. H. Hubel and T. N. Wiesel, "Receptive fields and functional architecture of monkey striate cortex," *J. Physiol.*, vol. 195, no. 1, pp. 215–243, Mar. 1968.

[21] P. Roy, S. Ghosh, S. Bhattacharya, and U. Pal, "Effects of degradations on deep neural network architectures," 2018, *arXiv:1807.10108*. [Online]. Available: https://arxiv.org/abs/1807.10108

[22] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," 2017, *arXiv:1704.04861*. [Online]. Available: https://arxiv.org/abs/1704.04861

[23] P. Nair, R. Doshi, and S. Keselj, "Pushing the limits of capsule networks," 2021, *arxiv:2103.08074*. [Online]. Available: https://arxiv.org/abs/2103.08074

[24] T. Hahn, M. Pyeon, and G. Kim, "Self-routing capsule networks," in *Proc. 33rd Conf. Neural Inf. Process. Syst. (NeurIPS)*, 2019, pp. 1–13.

[25] I. Paik, T. Kwak, and I. Kim, "Capsule networks need an improved routing algorithm," in *Proc. 11th Asian Conf. Mach. Learn.*, 2019, pp. 489–502.

[26] J. Gu and V. Tresp, "Improving the robustness of capsule networks to image affine transformations," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2020, pp. 7283–7291.

[27] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-MNIST: A novel image dataset for benchmarking machine learning algorithms," 2017, *arXiv:1708.07747*. [Online]. Available: https://arxiv.org/abs/1708.07747

[28] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng, "Reading digits in natural images with unsupervised feature learning," in *Proc. NIPS Workshop Deep Learn. Unsupervised Feature Learn.*, 2011, pp. 1–9.

[29] A. Krizhevsky *et al.*, "Learning multiple layers of features from tiny images," Citeseer, Tech. Rep., 2009

[30] J. Howard. (2018). *Imagenette*. [Online]. Available: https://github.com/fastai/imagenette/

[31] L. Bossard, M. Guillaumin, and L. Van Gool, "Food-101–mining discriminative components with random forests," in *Proc. Eur. Conf. Comput. Vis.*, 2014, pp. 446–461.

[32] Z. Chen and D. Crandall, "Generalized capsule networks with trainable routing procedure," 2018, *arXiv:1808.08692*. [Online]. Available: https://arxiv.org/abs/1808.08692

[33] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proc. Int. Conf. Mach. Learn.*, 2015, pp. 448–456.

[34] P. Izmailov, D. Podoprikhin, T. Garipov, D. P. Vetrov, and A. G. Wilson, "Averaging weights leads to wider optima and better generalization," in *Proc. 34th Conf. Uncertainty Artif. Intell. (UAI)*, 2018, pp. 1–12.

[35] M. Lin, Q. Chen, and S. Yan, "Network in network," 2013, *arXiv:1312.4400*. [Online]. Available: https://arxiv.org/abs/1312.4400

[36] K. Ahmed and L. Torresani, "STAR-CAPS: Capsule networks with straight-through attentive routing," in *Proc. 33rd Conf. Neural Inf. Process. Syst. (NeurIPS)*, 2019, pp. 9098–9107.

ADAM BYERLY received the M.S. degree in computer science from Bradley University, Peoria, IL, USA, in 2015. He is currently pursuing the Ph.D. degree with the Department of Electronic and Electrical Engineering, Brunel University London, Uxbridge, U.K.

He is also lectures with the Department of Computer Science and Information Systems, Bradley University, Peoria, IL. Prior to transitioning to academia, he accumulated over 20 years of industry experience in software engineering and architecture in various industries, including marketing, financial, insurance, and manufacturing.

TATIANA KALGANOVA received the B.Sc. (Hons.) and Ph.D. degrees.

She is currently a Reader in intelligent systems and the ECE Postgraduate Research Director in ECE with Brunel University London, Uxbridge, U.K. She has over 20 years of experience in design and implementation of applied intelligent systems.

● ● ●