Path Capsule Networks

Mohammed Amer¹ · Tomás Maul¹

Published online: 28 May 2020
© Springer Science+Business Media, LLC, part of Springer Nature 2020

Abstract
Capsule network (CapsNet) was introduced as an enhancement over convolutional neural networks, supplementing the latter’s invariance properties with equivariance through pose estimation. CapsNet achieved a very decent performance with a shallow architecture and a significant reduction in parameters count. However, the width of the first layer in CapsNet is still contributing to a significant number of its parameters and the shallowness may be limiting the representational power of the capsules. To address these limitations, we introduce Path Capsule Network (PathCapsNet), a deep parallel multi-path version of CapsNet. We show that a judicious coordination of depth, max-pooling, regularization by DropCircuit and a new fan-in routing by agreement technique can achieve better or comparable results to CapsNet, while further reducing the parameter count significantly.

Keywords Neural networks · Capsule networks · Modularity

1 Introduction
Convolutional Neural Networks (CNNs) [4,17] have remained state-of-the-art in image processing and computer vision tasks since their successful large scale training by Krizhevsky et al. [14]. CNNs were biologically inspired by the visual cortex [13] and were built on the principle of translation invariance, achieved through local receptive fields, weight sharing and pooling operations. Despite their success, CNNs suffer from inherent limitations, most significantly the fact that translation invariance by definition causes loss of location information. This limitation has stimulated a lot of research in the direction of augmenting learning with location data [5,29,31].

Sabour et al. [24] argued that the main limitation of CNNs is the focus on achieving translation invariance, and that equivariance should also be targeted. Hence, the authors proposed CapsNet as a step towards achieving equivariance. The philosophy of CapsNet is that a single activation/feature should be replaced by a pose vector, named capsule, representing the different properties of an object’s viewpoint. CapsNet has two main components, which are

¹ School of Computer Science, University of Nottingham, Semenyih, Malaysia
PrimaryCapsule and DigitCaps layers. PrimaryCapsules represent the different parts of the underlying objects, which are then multiplied by translation matrices to get prediction vectors, representing the votes of each PrimaryCapsule with respect to each DigitCaps, which are then routed using routing by agreement to compute DigitCaps activations, which can then be used to signify the presence of an object. The philosophy is that with changing the viewpoint of an object, the change in pose matrices should be coordinated, such that the voting agreement is maintained. We consider using another form of routing by agreement, fan-in routing in contrast to fan-out routing used by Sabour et al. [24], which we show can have better performance under some conditions.

CapsNet was shown to achieve very good results with a shallow architecture and decent parameter savings, compared with deep CNNs. However, the lack of depth can be limiting to the expressiveness of the network. Moreover, the first convolutional layer in CapsNet is large and contributes to increasing the number of CapsNet parameters significantly. We believe that a coordinated inclusion of depth and multiple pathways can help increase the network performance and simultaneously help save more parameters.

We consider a multipath architecture for including more depth into CapsNet. Multiple paths in neural networks are biologically plausible and biological neural networks have been shown to exhibit multipath parallel processing [6,20]. Aside from biological inspiration, we think that using different paths for generating PrimaryCapsules can be exploited to enhance performance while saving parameters significantly. A PrimaryCapsule generated by a deep path can be considered a deep version of the original CapsNet capsule, which we believe can exhibit more expressiveness and more abstraction, similar to other deep structures in the deep learning paradigm.

The universal approximation theorem by Hornik et al. [12] showed that any Borel measurable function can be approximated by a sufficiently wide single layer multilayer perceptron (MLP). Empirically, however, this is infeasible due to optimization limitations, and is rarely desirable due to the problem of overfitting. On the other hand, making use of depth is statistically motivated by composition of functions and empirically can lead to better generalization. Moreover, as we show, depth can be added judiciously to save parameters without sacrificing performance.

Our contributions in this paper are:

1. We propose PathCapsNet, a multipath deep version of CapsNet.
2. We enrich the routing by agreement methodology by a new variant, fan-in routing.
3. By carefully adding depth and max-pooling, along with a multi-path structure, fan-in routing and DropCircuit, we achieved better or comparable results to CapsNet with significant parameter savings.
4. We open the possibility of leveraging significant model parallelism in the context of capsule networks.

In the next section we discuss the previous work done around capsule networks and multipath architectures, and how we enhance by building on these concepts.

2 Related Work

CapsNet [24] was introduced as an architecture that builds up on the conventional CNN [4,16] trying to overcome its limitations. The main motivation behind CapsNet is achieving equivariance, in addition to the invariance properties already implemented by CNN. CapsNet could achieve a good generalization using relatively fewer parameters than deep CNNs (only
8.2M parameters for the MNIST model with reconstruction). Different variants have been introduced since the original CapsNet. Phaye et al. [23] introduced DCNet as a dense version of capsule networks and DCNet++ by stacking multiple DCNets. In DCNet++, each DCNet in the stack produces its version of the PrimaryCapsule layer, which is then fed to the next DCNet in the stack. The final output is calculated based on both the output of each subnetwork and their concatenation. They also made some modifications to the decoder (reconstruction) subnetwork. DCNet++ achieved good generalization in relatively few epochs at the cost of using more parameters (13.4M).

Another variant is MS-CapsNet [32]. MS-CapsNet is composed of three successive modules. The first module is the feature extractor and it has two convolutional paths of depths 1 and 2 and a third path which is just a skip connection. Each path produces a PrimaryCapsule of different dimension. The second module is a capsule encoding and it is responsible for projecting the PrimaryCapsules to a common dimension and concatenating them. The third module, capsule dropout, is applied before routing and it is responsible for dropping random capsules as a way of regularization in a manner similar to dropout [25] and other similar techniques. Capsule dropout showed enhancement in performance relative to the non-dropout condition. MS-CapsNet could achieve better performance than the original CapsNet on FashionMNIST and CIFAR-10 with fewer parameters (~11M).

SECaps [7] is an adaptation of CapsNet to sequential tasks, specifically Natural Language Processing (NLP). The word embeddings of single words are treated as PrimaryCapsules. Since the dynamic routing is not sequential in nature and doesn’t respect order, the seq-caps layer is introduced. This layer is basically composed of a long short-term memory (LSTM) layer that is applied to a given sequence of the data as a series encoding, and then the output is dynamically routed in the conventional way to produce the output of the next layer. Multiple seq-caps layers can be stacked. Another module, the attention module, transforms the word embeddings, which are then concatenated with the seq-layer output. The final output is produced by an MLP subnetwork. SECaps was evaluated on multiple charge prediction datasets, achieving better performance than the state-of-the-art.

Siamese capsule network (SCN) [19] is the capsule version of the conventional siamese network. Neill et al. [19] introduced SCN as a face verification approach similar to DeepFace [28]. SCN is very similar in architecture to the original CapsNet. It has a convolutional layer, followed by the PrimaryCapsules layer and then a layer called Face Capsule layer, which is essentially similar to the DigitCaps layer. The final output is produced by a fully connected layer on top of the Face Capsule layer. SCN achieved good performance on different datasets with a smaller model, little preprocessing and less data.

Matrix capsules network was proposed by Hinton et al. [9] as a generalization of the original CapsNet for more efficient pose estimation. Each capsule is represented by a matrix and a sigmoid unit that controls the probability of activating the capsule. Every pose matrix is multiplied by a transformation matrix to get the votes which will be used for routing to the next layer. Routing is done using expectation maximization (EM) that takes as input the votes and activation probabilities of the previous layer. Matrix capsules network achieved a very good accuracy improvement on the smallNORB dataset, a dataset that is highly viewpoint variant, but it seems that it doesn’t have the same advantage on MNIST.

The ideas of branching, parallel computation and multiple paths are well established in the deep learning literature and have their supporting biological plausibility [6,20]. In Cireşan et al. [3], each path in a multi-path CNN is trained on a different preprocessing/distortion of the input image and the columns outputs are averaged to produce the final output. A similar approach is used in Wang [30], but with different types of inputs which are the source image and a bilateral filtered version of it, and the outputs of the paths are integrated using
fully connected layers. Similarly, in Yu et al. [34] and in the context of image captioning, multi-view feature learning is achieved using several detectors applied in parallel. The same idea of parallel detectors is also used in Zhang et al. [36] in the context of unsupervised dimensionality reduction. Zhang et al. [36] use a set of local contractive autoencoders to extract representations of the neighbourhood of each sample, and then apply local affine transforms to align the local features with a global coordinate system. For extracting hierarchical features in the context of predicting the click feature vector of images, Yu et al. [35] apply a parallel structure to extract features of different granularity from the word and image modalities. In the context of HPR (human pose recovery), Hong et al. [10,11] utilize parallel structures in different ways. Hong et al. [10] use two parallel autoencoders to extract 2D and 3D from different modalities and then learn the mapping from 2D to 3D features. On the other hand, Hong et al. [11] use a model that takes different modalities as inputs and then produce different outputs using a branching structure, where each output corresponds to a different reconstruction task of a specific modality.

Szegedy et al. [26] proposed the Inception-v1 model, which was responsible for winning ILSVRC-14, and is composed of a highly branched multipath architecture. Szegedy et al. [27] further improved the design of Inception-v1 to produce Inception-v2&3 which exploit large scale branching and multiple paths even more. The Xception architecture [2] is a further extension to the Inception family, that uses more branching based on separable convolutions. ResNetXt [33] and Residual Inception [37] are extensions of ResNet [8] where the modular block is multipath instead of single path. FractalNet [15] is another type of architecture that has a recursive self-similar, highly branched structure. Parallel circuit networks, introduced by Phan et al. [21], adopt an extensively multipath architecture, and have demonstrated generalization improvements using a dropping technique called DropCircuit [22]. Related to the DropCircuit technique is the path dropout used by Bender et al. [1] to regularize the training of a one-shot model, which is an implicit form of a multipath network, where a whole space of possible branches is trained simultaneously.

We build on previous work by:

1. Adding representational power to PrimaryCapsules by generating each capsule using a deep path.
2. Enriching dynamic routing by agreement with a new fan-in variant.
3. Combining depth, a multipath architecture, DropCircuit, max-pooling and fan-in routing to obtain a level of performance congruent with the original CapsNet, with significant parameter savings.
4. Showing that max-pooling is not inherently contradictory with the CapsNet philosophy, and that it can be used to save parameters significantly without sacrificing neither performance nor pose awareness.
5. Opening a new illuminating perspective on the potential diversity of the representations learned by multipath architectures using RSA-analysis [18].

In the next section, we explain the general PathCapsNet architecture and the different pieces that contribute to its performance.

### 3 Methods

The original CapsNet [24] has two main capsule types, namely the PrimaryCapsules and the DigitCaps. PrimaryCapsules are formed by applying an initial convolution layer to produce 256 channels, then another set of convolutions, which are then rearranged into 32 8D Pri-
maryCapsules. PrimaryCapsules are then routed to the next DigitCaps layer using dynamic routing by agreement. In one variant of CapsNet, namely CapsNet with reconstruction, a reconstruction layer is learned on top of the DigitCaps layer to facilitate the learning of instantiation (or transformation) parameters and therefore enhance generalization.

PathCapsNet Fig. 1 shares the upper part of CapsNet, starting from the PrimaryCapsules layer, through the DigitCaps layer and ending with a reconstruction layer if needed. However, PathCapsNet is fundamentally different in how the PrimaryCapsules are constructed. In PathCapsNet, each PrimaryCapsule is formed by a deep CNN, named a path. So, the input is fed into different CNNs (paths) and the output of each path comprises one PrimaryCapsule.

The experiments done by Phan et al. [22] demonstrate enhanced generalization in multi-path MLPs, named parallel circuits in their work, using a drop technique called DropCircuit. DropCircuit is an adaptation of dropout to multipath architectures, where different paths are dropped during training, using a pre-specified probability. This is believed to enhance generalization by promoting independence between paths, hence allowing for problem decomposition and learning more useful representations, similar to dropout [25] and related techniques.

Dynamic routing is the mechanism by which PrimaryCapsules are routed to DigitCaps capsules, such that similar votes from PrimaryCapsules contribute more strongly to the target
DigitCaps. The dynamic routing by agreement algorithm used in Sabour et al. [24] updates the contribution of votes based on the similarity between the output DigitCaps and the prediction vector, representing the vote, using dot product as a measure of similarity. So, given the prediction vectors (votes) from the previous layer of capsules (PrimaryCapsule layer) $\hat{u}_{ji}$, where $j$ is the index of the DigitCaps capsule and $i$ is the index of a single capsule in the PrimaryCapsule layer, the output vector (DigitCaps) is calculated as,

$$s_j = \sum_i c_{ij} \hat{u}_{ji}$$  \hspace{1cm} (1)

where $c_{ij}$ are the coupling coefficients weighting the contributions of different prediction vectors,

$$c_{ij}^{(f_{out})} = \frac{\exp(b_{ij})}{\sum_k \exp(b_{ik})}$$  \hspace{1cm} (2)

and $b_{ij}$ is the log probability (logits) that the $i$th PrimaryCapsule should be coupled to the $j$th DigitCaps capsule. We call this fan-out (fout) routing, since the weights of the contributions of the $i$th PrimaryCapsule to each DigitCaps capsule in the next layer are normalized probabilities that sum to 1.0. For PathCapsNet, we used a different form of dynamic routing by agreement, named fan-in (fin) routing, where logits are normalized such that the weights of the contributions to the $j$th DigitCaps capsule from all the PrimaryCapsules are normalized probabilities that sum to 1.0. Coupling coefficients for fan-in routing are calculated as,

$$c_{ij}^{(f_{in})} = \frac{\exp(b_{ij})}{\sum_k \exp(b_{ki})}$$  \hspace{1cm} (3)

Figure 2 shows a simplified diagram highlighting the difference in Softmax calculation direction between fan-out and fan-in variants. All connections with similar line pattern are inputs to the same Softmax. Note how the Softmax is applied across connections fanning into the same DigitsCaps in the fan-in variant, while it is applied across connections fanning out from a single PrimaryCapsule in the fan-out variant.

---

Fig. 2 A simplified diagram showing the main difference in Softmax direction between fan-out and fan-in routing. Connections with similar lines are inputs to the same Softmax. Note that the other operations between PrimaryCapsules and DigitsCaps layers are abridged for clarity.
Table 1 Single path architecture

| Layer | Type    | Kernel | Padding | Stride | Output channels |
|-------|---------|--------|---------|--------|-----------------|
| 1     | Conv    | 9      | 4       | 1      | 16              |
| 2     | Conv    | 9      | 4       | 1      | 16              |
| 3     | Maxpool | 2      | 0       | 2      | 16              |
| 4     | Conv    | 9      | 4       | 1      | 16              |
| 5     | Conv    | 9      | 4       | 1      | 8               |
| 6     | Maxpool | 2      | 0       | 2      | 8               |

The accuracy and reconstruction losses are calculated the same way as Sabour et al. [24], using margin loss and sum of squared errors loss, respectively.

In the next section, we present the details of our experimental design and the results we obtained.

4 Results

4.1 PathCapsNet Architecture

For all of our experiments, each path had the same architecture Table 1. The number of paths in each experiment varied and will be clarified for each set of experimental results. We will use the notation PathCapsNet-[num], where [num] is replaced by the number of paths, so PathCapsNet-5 is PathCapsNet with 5 paths. All PrimaryCapsules were 8D with spatial dimensions $7 \times 7$. As each path produces one PrimaryCapsule, the number of PrimaryCapsules is equal to the number of paths. The DigitCaps layer was exactly the same as Sabour et al. [24]. We used 3 routing iterations in all the experiments and whenever we used fan-in routing, we initialized the transformation matrices of the DigitCaps layer randomly from a standard normal distribution. The Adam optimizer was used in all of the experiments using the default parameters and learning rate. When DropCircuit was used, the probability of path dropping was 0.5. Our benchmark was the original CapsNet [24] with and without reconstruction and using 3 routing iterations. The benchmark was implemented using the same architecture as reported in the original paper without any modifications, unless otherwise specified. All reported results are based on an average of three trials, with 300 epochs of training each. This number of epochs is relatively small compared to the number used by Sabour et al. [24], which seems to be more than 1000 epochs.

The following 3 sub-sections will describe the results of our experiments, applying the mentioned model architecture on 3 different datasets, namely, MNIST, CIFAR10 and iWildCam2019. In the tables summarizing the results, the best test performance across all conditions was bolded and we marked the best test performance across PathCapsNet conditions by an asterisk. The fourth and last sub-section in the results, will present an analytical perspective on the learning characteristics of the proposed PathCapsNet model compared to CapsNet.
Table 2  MNIST results

| No. | Architecture   | Routing | Paths | DropCircuit | Parameters count | Parameters (%) | Test error (%) |
|-----|----------------|---------|-------|-------------|------------------|----------------|----------------|
| 1   | CapsNet        | Fan-out | N/A   | N/A         | 6.8M             | 100            | 0.48 ± 0.02    |
| 2   | CapsNet        | Fan-in  | N/A   | N/A         | 6.8M             | 100            | 0.42 ± 0.03    |
| 3   | PathCapsNet    | Fan-out | 5     | Yes         | 683K             | 10             | 0.54 ± 0.05    |
| 4   | PathCapsNet    | Fan-in  | 5     | No          | 683K             | 10             | 0.48 ± 0.07    |
| 5   | PathCapsNet    | Fan-in  | 5     | Yes         | 683K             | 10             | 0.47 ± 0.04    |
| 6   | PathCapsNet    | Fan-in  | 10    | No          | 1.4M             | 21             | 0.52 ± 0.03    |
| 7   | PathCapsNet    | Fan-in  | 10    | Yes         | 1.4M             | 21             | *0.42 ± 0.05   |

Reconstruction

| No. | Architecture   | Routing | Paths | DropCircuit | Parameters count | Parameters (%) | Test error (%) |
|-----|----------------|---------|-------|-------------|------------------|----------------|----------------|
| 1   | CapsNet        | Fan-out | N/A   | N/A         | 8.2M             | 100            | 0.35 ± 0.04    |
| 2   | CapsNet        | Fan-in  | N/A   | N/A         | 8.2M             | 100            | 0.47 ± 0.03    |
| 3   | PathCapsNet    | Fan-out | 10    | No          | 2.8M             | 34             | 0.44 ± 0.06    |
| 4   | PathCapsNet    | Fan-in  | 10    | No          | 2.8M             | 34             | 0.47 ± 0.02    |
| 5   | PathCapsNet    | Fan-out | 10    | Yes         | 2.8M             | 34             | 0.49 ± 0.02    |
| 6   | PathCapsNet    | Fan-in  | 10    | Yes         | 2.8M             | 34             | 0.42 ± 0.05    |
| 7   | PathCapsNet    | Fan-in  | 16    | Yes         | 3.6M             | 44             | *0.38 ± 0.02   |

| No reconstruction |

| No. | Architecture   | Routing | Paths | DropCircuit | Parameters count | Parameters (%) | Test error (%) |
|-----|----------------|---------|-------|-------------|------------------|----------------|----------------|
| 1   | CapsNet        | Fan-out | N/A   | N/A         | 6.8M             | 100            | 0.48 ± 0.02    |
| 2   | CapsNet        | Fan-in  | N/A   | N/A         | 6.8M             | 100            | 0.42 ± 0.03    |
| 3   | PathCapsNet    | Fan-out | 5     | Yes         | 683K             | 10             | 0.54 ± 0.05    |
| 4   | PathCapsNet    | Fan-in  | 5     | No          | 683K             | 10             | 0.48 ± 0.07    |
| 5   | PathCapsNet    | Fan-in  | 5     | Yes         | 683K             | 10             | 0.47 ± 0.04    |
| 6   | PathCapsNet    | Fan-in  | 10    | No          | 1.4M             | 21             | 0.52 ± 0.03    |
| 7   | PathCapsNet    | Fan-in  | 10    | Yes         | 1.4M             | 21             | *0.42 ± 0.05   |

Reconstruction

| No. | Architecture   | Routing | Paths | DropCircuit | Parameters count | Parameters (%) | Test error (%) |
|-----|----------------|---------|-------|-------------|------------------|----------------|----------------|
| 1   | CapsNet        | Fan-out | N/A   | N/A         | 8.2M             | 100            | 0.35 ± 0.04    |
| 2   | CapsNet        | Fan-in  | N/A   | N/A         | 8.2M             | 100            | 0.47 ± 0.03    |
| 3   | PathCapsNet    | Fan-out | 10    | No          | 2.8M             | 34             | 0.44 ± 0.06    |
| 4   | PathCapsNet    | Fan-in  | 10    | No          | 2.8M             | 34             | 0.47 ± 0.02    |
| 5   | PathCapsNet    | Fan-out | 10    | Yes         | 2.8M             | 34             | 0.49 ± 0.02    |
| 6   | PathCapsNet    | Fan-in  | 10    | Yes         | 2.8M             | 34             | 0.42 ± 0.05    |
| 7   | PathCapsNet    | Fan-in  | 16    | Yes         | 3.6M             | 44             | *0.38 ± 0.02   |

Bold values indicate the best performance across all conditions
*Best performance across PathCapsNet conditions

4.2 MNIST

For MNIST, we trained on 90% of the training dataset and left 10% for calculating validation performance, used for selecting the best model. The test performance was reported on the full test set. Following Sabour et al. [24], the only augmentation used during training was padding by 2 and random cropping using a 28 × 28 patch. Our performance results on MNIST are summarized in Table 2.

For the no-reconstruction setting, fan-in routing improved CapsNet test performance from 0.48 to 0.42%. A similar improvement was observed for PathCapsNet-5, where test performance improved from 0.54 to 0.47%. With DropCircuit, we observed no improvement for a small number of paths, i.e PathCapsNet-5, while a significant improvement was observed for PathCapsNet-10, where the error improved from 0.52 to 0.42%, which had the same performance as the standard CapsNet with only 21% of the parameters. A regularization effect can be observed from the validation curves Fig. 3. For the reconstruction setting, both of CapsNet and PathCapsNet-16 with fan-in and DropCircuit had the best test performance, with PathCapsNet-16 having only 44% of the parameters. Combining DropCircuit and fan-out had the same or worse performance compared to the corresponding no-DropCircuit condition, everything else being equal. The reverse seemed to hold for DropCircuit when combined with fan-in, where the performance was better or the same.

4.3 CIFAR10

For CIFAR10, we used the same split scheme of MNIST, i.e 90% for training and 10% for validation. Also, the test error was reported, in the same way, on the full test set. No
Fig. 3  Average validation curves for PathCapsNet-10 with fan-in routing (MNIST)

Table 3  CIFAR10 results

| No. | Architecture | Routing | Paths | DropCircuit | Parameters count | Parameters (%) | Test error (%) |
|-----|--------------|---------|-------|-------------|------------------|----------------|----------------|
| No reconstruction |
| 1 | CapsNet | Fan-out | N/A | N/A | 8.0M | 100 | 35.6 ± 0.4 |
| 1 | CapsNet | Fan-in | N/A | N/A | 8.0M | 100 | 32.7 ± 1.1 |
| 1 | PathCapsNet | Fan-out | 16 | No | 2.2M | 28 | 31.3 ± 1.2 |
| 1 | PathCapsNet | Fan-in | 16 | No | 2.2M | 28 | 31.7 ± 0.3 |
| 1 | PathCapsNet | Fan-out | 16 | Yes | 2.2M | 28 | 35.6 ± 1.4 |
| 1 | PathCapsNet | Fan-in | 16 | Yes | 2.2M | 28 | *30.6 ± 0.7 |
| Reconstruction |
| 1 | CapsNet | Fan-out | N/A | N/A | 11.7M | 100 | 35.5 ± 0.8 |
| 1 | CapsNet | Fan-in | N/A | N/A | 11.7M | 100 | 32.9 ± 1.2 |
| 1 | PathCapsNet | Fan-out | 16 | No | 6.0M | 51 | 30.9 ± 0.4 |
| 1 | PathCapsNet | Fan-in | 16 | No | 6.0M | 51 | 32.2 ± 0.9 |
| 1 | PathCapsNet | Fan-out | 16 | Yes | 6.0M | 51 | 35.2 ± 1.5 |
| 1 | PathCapsNet | Fan-in | 16 | Yes | 6.0M | 51 | *30.7 ± 0.7 |

Bold values indicate the best performance across all conditions
*Best performance across PathCapsNet conditions

augmentation was used in this dataset. All the PathCapsNet conditions for CIFAR10 had 16 paths. CIFAR10 results are summarized in Table 3.

For both of the no-reconstruction and reconstruction conditions, fan-in routing improved CapsNet performance. All PathCapsNet conditions, except PathCapsNet with fan-out and DropCircuit, were better than CapsNet. The best performance was achieved by PathCapsNet with fan-in and DropCircuit, which was better than CapsNet by around 5%, using only 28% and 51% of the CapsNet parameters in the no-reconstruction and reconstruction settings, respectively. For all conditions, everything else being equal, adding DropCircuit to PathCapsNet fan-out models worsened the performance, while it improved the performance for fan-in models.
Table 4  iWildCam2019 results

| No. | Architecture | Routing | Paths | DropCircuit | Parameters count | Parameters (%) | Test error (%) |
|-----|--------------|---------|-------|-------------|------------------|----------------|---------------|
| 1   | CapsNet      | Fan-out | N/A   | N/A         | 6.6M             | 100            | 21.6 ± 0.8   |
| 1   | CapsNet      | Fan-in  | N/A   | N/A         | 6.6M             | 100            | 23.8 ± 0.3   |
| 1   | PathCapsNet  | Fan-out | 16    | No          | 1.7M             | 26             | 22.5 ± 0.5   |
| 1   | PathCapsNet  | Fan-in  | 16    | No          | 1.7M             | 26             | 24.3 ± 1.6   |
| 1   | PathCapsNet  | Fan-out | 16    | Yes         | 1.7M             | 26             | *21.8 ± 1.1  |
| 1   | PathCapsNet  | Fan-in  | 16    | Yes         | 1.7M             | 26             | 24.3 ± 0.5   |

Reconstruction

| 1   | CapsNet      | Fan-out | N/A   | N/A         | 8.0M             | 100            | 21.6 ± 0.6   |
| 1   | CapsNet      | Fan-in  | N/A   | N/A         | 8.0M             | 100            | 23.1 ± 0.3   |
| 1   | PathCapsNet  | Fan-out | 16    | No          | 3.0M             | 38             | 22.1 ± 1.0   |
| 1   | PathCapsNet  | Fan-in  | 16    | No          | 3.0M             | 38             | 22.3 ± 0.6   |
| 1   | PathCapsNet  | Fan-out | 16    | Yes         | 3.0M             | 38             | *21.9 ± 0.9  |
| 1   | PathCapsNet  | Fan-in  | 16    | Yes         | 3.0M             | 38             | 24.1 ± 1.0   |

Bold values indicate the best performance across all conditions
*Best performance across PathCapsNet conditions

4.4 iWildCam2019

iWildCam2019 is a camera trap dataset composed of images of labeled animal species. The original dataset has a training set of 196,157 images, a test set of 153,730 images and 23 labels (22 species and an empty label). Since there is no complete overlap between the training and test set classes, we used only the original training set by splitting it into train, validation and test sets. This was done by first choosing 10 classes (deer, squirrel, rodent, fox, coyote, raccoon, skunk, cat, dog, opossum), then balancing all the classes by choosing only 1000 samples from each class. We then split the data into 70% train, 20% validation and 10% test. This was done class-wise to maintain the balance. The samples were converted to grey-scale and downsampled to $23 \times 32$ pixels. No augmentation was used. All the PathCapsNet models had 16 paths. The results are summarized in Table 4.

For both of the no-reconstruction and reconstruction settings, and in contrast to the other datasets, CapsNet with fan-out was better than the corresponding fan-in condition. Combining DropCircuit with fan-out had the same or better performance compared to the corresponding no-DropCircuit condition. The reverse was true for fan-in. Both CapsNet with fan-out and PathCapsNet with fan-out and DropCircuit had the best test performance with PathCapsNet having only 26% and 38% of the CapsNet parameters in the no-reconstruction and reconstruction settings, respectively.

4.5 RSA Analysis

The main crucial difference between CapsNet and PathCapsNet is how the PrimaryCapsules are generated. In order to peek into the characteristics of the learned features at the PrimaryCapsules layer, we followed the RSA-based (Representational Similarity Analysis) methods used in Mehrer et al. [18]. For a given model, either CapsNet or PathCapsNet, we calculate the RDMs (Representational Dissimilarity Matrices) for each PrimaryCapsule.
RDMs are calculated by presenting pairs of different samples to a given model (or a part of a model) and then calculating a measure of distance between the corresponding outputs. We use Euclidean distance as our distance measure. We then calculate the squared Pearson correlation ($r^2$) between each pair of the PrimaryCapsules using the RDMs. After taking the average of $r^2$ across PrimaryCapsules pairs, we use it as a measure of the correlation between PrimaryCapsules in the given model. We use a subset of 500 samples from the test set of each dataset for calculating the RDMs and the reported correlation for each model architecture is an average of three independently trained models.

For the MNIST dataset Fig. 4, CapsNet models showed a consistent high correlation between PrimaryCapsules. In contrast, PathCapsNet conditions didn’t show a consistent pattern of correlation. Some PathCapsNet conditions showed very low correlation, while others showed high correlation, sometimes higher than CapsNet. For CIFAR10 and iWildCam2019 Figs. 5 and 6, CapsNet still showed a consistent high correlation. However, in contrast to MNIST, all the PathCapsNet conditions showed a consistent low correlation.
In the next section we discuss our interpretations and hypotheses explaining the different techniques that contributed to these results and why we think that they enhance the current methodologies.

5 Discussion

Three main components, we believe, contributed to the performance of PathCapsNet, namely deep PrimaryCapsules, fan-in routing and DropCircuit. Deep paths, even without fan-in routing and no DropCircuit, could achieve a better or near test performance. We attribute this to the increased representational power of each PrimaryCapsule.

The main rationale behind fan-out routing was that each detected part of an object should contribute more strongly to a single object category rather than to multiple object categories. Fan-in routing, on the other hand, is expressing the idea that for a given object, different detected parts should contribute differently. Both philosophies can be seen to have different pros and cons, making each one optimal for a different set of contexts. This is supported empirically since fan-in achieved better performance on average in MNIST and CIFAR10, while fan-out was mostly better in iWildCam2019.

DropCircuit showed performance enhancement for conditions with large numbers of paths, and no significant effect for small numbers of paths. We believe DropCircuit, being a form of regularization, needs a sufficiently large number of paths to show a positive effect. With a small number of paths, dropping becomes too destructive, specially with a high dropping rate, to show any significant improvement. We believe DropCircuit, like other drop techniques, is introducing independence between paths and promoting the extraction of more useful PrimaryCapsule representations. This is partly supported by the RSA-analysis, which is further discussed later in this section.

A mutual interaction seems to exist between the routing mode (i.e fan-out and fan-in) and DropCircuit. For MNIST and CIFAR10, where fan-in seems to improve performance over fan-out, DropCircuit enhances the performance of fan-in models, while it worsens fan-out models. For iWildCam2019, where fan-out is mostly better than fan-in, DropCircuit seems
We hypothesize that a contributing factor to PathCapsNet performance is the ability of the independent paths to extract diverse uncorrelated representations. This, however, seems to depend not only on the model architecture, but also on the dataset structure. RSA-analysis showed that while CapsNet had a consistent high correlation between its PrimaryCapsules in all the datasets, PathCapsNet showed a consistent low correlation in CIFAR10 and iWildCam2019. However, this seems, as mentioned, to be dataset dependent since in MNIST, and despite the fact that some PathCapsNet models showed low correlation, others showed higher correlation than CapsNet. Since this high PathCapsNet correlation seems to happen with DropCircuit, one potential explanation is that due to the similarity between MNIST classes, learning diverse features is difficult and enforcing independent learning through DropCircuit leads only to convergence of the learned features.

We could achieve a performance comparable to CapsNet with significant parameter savings. This was possible thanks to a careful coordination of depth, multi-path structure and regularization. Essentially, we substituted the wide convolutional layer of CapsNet with deeper narrower paths regularized by DropCircuit. Fan-in routing alone enhanced the performance of two out of the three datasets used. Moreover, it enabled the effective utilization of DropCircuit regularization in these cases, since, as discussed, the DropCircuit effect depends on the routing mode. Another component that contributed to reducing parameter counts was max-pooling which may be considered incompatible with CapsNet and its equivariance aim. However, we showed that it is possible to use max-pooling layers in PathCapsNet, which allowed further parameter savings without sacrificing performance.

Moreover, experiments perturbing different dimensions in the DigitCaps layer learned on MNIST confirmed that, even when using max-pooling, different pose parameters can be successfully learned Fig. 7. For example, perturbing the first three dimensions of the DigitCaps layer of one of the models Fig. 7 suggested that the first dimension in the model was controlling multiple pose parameters, like the elongation of the circular and linear regions and stroke thickness. The second dimension seemed to affect the circle axes orientations and also stroke thickness, while the third dimension resulted in a combination of vertical translation, vertical axis inclination and elongation of the circular part.
6 Conclusion

We have introduced PathCapsNet, a multipath capsule network that can achieve better or comparable performance to CapsNet with significant parameter savings. In order to achieve this, we used regularization by DropCircuit along with a new variant of dynamic routing by agreement, fan-in routing. The careful coordination of depth, max-pooling, routing mode and DropCircuit allowed for maintaining CapsNet performance, while cutting down parameter counts considerably. RSA-analysis suggested that the independent multipath architecture of PathCapsNet has a diversifying effect on the learned representations. Reconstructions with perturbations showed that the use of max-pooling is not necessarily in conflict with retaining location information and pose estimation. The independence of paths renders the model suitable for model parallelism, a property which we didn’t investigate in detail and we leave for future work. We think there is still more space for enhancing PathCapsNet, specially in the reconstruction setting where we believe there is a complex interaction between routing, DropCircuit and reconstruction.

Acknowledgements  We acknowledge the use of Athena at HPC Midlands++, which was funded by the EPSRC on Grant EP/P020232/1, in this research, as part of the HPC Midlands+ consortium. This work was partially supported by a Grant from Microsoft’s AI for Earth program.

References

1. Bender G, Kindermans PJ, Zoph B, Vasudevan V, Le Q (2018) Understanding and simplifying one-shot architecture search. http://proceedings.mlr.press/v80/bender18a
2. Chollet F (2016) Xception: deep learning with depthwise separable convolutions. arXiv:1610.02357
3. Cireşan D, Meier U, Schmidhuber J (2012) Multi-column deep neural networks for image classification arXiv:1204.2800
4. Fukushima K, Miyake S (1980) Neocognitron: Self-organizing network capable of position-invariant recognition of patterns. In: Proceedings of the 5th international conference pattern recognition, vol 1, pp 459–461
5. Ghafoorian M, Karssemeijer N, Heskes T, van Uden IWM, Sanchez CI, Liitens G, de Leeuw FE, van Ginneken B (2017) Location sensitive deep convolutional neural networks for segmentation of white matter hyperintensities. Sci Rep 7(1):5110.https://doi.org/10.1038/s41598-017-05300-5
6. Gollisch T, Meister M (2010) Eye smarter than scientists believed: neural computations in circuits of the retina. Neuron 65(2):150–64. https://doi.org/10.1016/j.neuron.2009.12.009
7. He C, Peng L, Le Y, He J (2018) SECaps: a sequence enhanced capsule model for charge prediction. arXiv:1810.04465
8. He K, Zhang X, Ren S, Sun J (2016) Deep residual learning for image recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp 770–778
9. Hinton GE, Salakhutdinov R (2006) Reducing the dimensionality of data with neural networks. Science 313(5786):504–07. https://doi.org/10.1126/science.1127647
10. Hong C, Yu J, Wan J, Tao D, Wang M (2015) Multimodal deep autoencoder for human pose recovery. IEEE Trans Image Process 24(12):5659–5670. https://doi.org/10.1109/TIP.2015.2487860
11. Hong C, Yu J, Zhang J, Jin X, Lee KH (2019) Multimodal face-pose estimation with multitask manifold deep learning. IEEE Trans Ind Inf 15(7):3952–3961. https://doi.org/10.1109/TII.2018.2884211
12. Hornik K, Stinchcombe M, White H (1990) Universal approximation of a unknown mapping and its derivatives using multilayer feedforward networks. Neural Netw 3(5):551–560. https://doi.org/10.1016/0893-6080(90)90005-6
13. Hubel D, Wiesel T (1968) Receptive fields and functional architecture of monkey striate cortex. J Phys. https://doi.org/10.1113/jphysiol.1968.sp008455
14. Krizhevsky A, Sutskever I, Hinton GE (2012) ImageNet classification with deep convolutional neural networks. https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks
15. Larsson G, Maire M, Shakhnarovich G (2016) FractalNet: ultra-deep neural networks without residuals. arXiv:1605.07648

16. LeCun Y, Boser B, Denker JS, Henderson D, Howard RE, Hubbard W, Jackel LD (1989) Backpropagation applied to handwritten zip code recognition. Neural Comput 4:541–551. https://doi.org/10.1162/neco.1989.1.4.541

17. LeCun Y, Bengio Y et al. (1995) Convolutional networks for images, speech, and time series. In: The handbook of brain theory and neural networks, vol 3361(10)

18. Mehrer J, Spoerer CJ, Kriegeskorte N, Kietzmann TC (2020) Individual differences among deep neural network models. bioRxiv. https://doi.org/10.1101/2020.01.08.989828

19. Neill JO (2018) Siamese capsule networks. arXiv:1805.07242

20. Otsuna H, Shinomiya K, Ito K (2014) Parallel neural pathways in higher visual centers of the Drosophila brain that mediate wavelength-specific behavior. Front Neural Circuits 8:8. https://doi.org/10.3389/fncir.2014.00008

21. Phan KT, Maul TH, Vu TT, Kin LW (2016) Improving neural network generalization by combining parallel circuits with dropout. Comput Sci. https://doi.org/10.1007/978-3-319-46675-0. arXiv:1612.04970

22. Phan KT, Maul TH, Vu TT, Lai WK (2018) DropCircuit: a modular regularizer for parallel circuit networks. Neural Process Lett 47(3):841–858. https://doi.org/10.1007/s11063-017-9677-4

23. Phaye SSR, Sikka A, Dhall A, Bathula D (2018) Dense and diverse capsule networks: making the capsules learn better arXiv:1805.04001

24. Sabour S, Frost N, Hinton GE (2017) Dynamic routing between capsules. arXiv:1710.09829

25. Srivastava N, Hinton G, Krizhevsky A, Sutskever I, Salakhutdinov R (2014) Dropout: a simple way to prevent neural networks from overfitting. J Mach Learn Res 15:1929–1958. https://doi.org/10.1214/12-AOS1000. arXiv:1102.4807

26. Szegedy C, Liu W, Jia Y, Sermanet P, Reed S, Anguelov D, Erhan D, Vanhoucke V, Rabinovich A (2015) Going deeper with convolutions. In: Proceedings of the IEEE computer society conference on computer vision and pattern recognition, vol 07–12 June, pp 1–9. https://doi.org/10.1109/CVPR.2015.7298594. arXiv:1409.4842

27. Szegedy C, Vanhoucke V, Ioffe S, Shlens J, Wojna Z (2015) Rethinking the inception architecture for computer vision. arXiv:1512.00567

28. Taigman Y, Yang M, Ranzato M, Wolf L (2014) DeepFace: closing the gap to human-level performance in face verification. https://www.cv-foundation.org/openaccess/content_cvpr_2014/html/Taigman_DeepFace_Closing_the_2014_CVPR_paper.html

29. Tang K, Paluri M, Fei-Fei L, Fergus R, Bourdev L (2015) Improving image classification with location context. arXiv:1505.03873

30. Wang M (2015) Multi-path Convolutional neural networks for complex image classification. arXiv:1506.04701

31. Wang Z, Veksler O (2018) Location augmentation for CNN. arXiv:1807.07044

32. Xiang C, Zhang L, Tang Y, Zou W, Xu C (2018) MS-CapsNet: a novel multi-scale capsule network. IEEE Signal Process Lett 25(12):1850–1854. https://doi.org/10.1109/LSP.2018.2873892

33. Xie S, Girshick R, Dollár P, Tu Z, He K (2016) Aggregated residual transformations for deep neural networks. arXiv preprint arXiv:1611.04510

34. Yu J, Li J, Yu Z, Huang Q (2019) Multimodal transformer with multi-view visual representation for image captioning. IEEE Trans Circuits Syst Video Technol. https://doi.org/10.1109/tcsvt.2019.2947482. arXiv:1905.07841

35. Yu J, Tan M, Zhang H, Tao D, Rui Y (2019) Hierarchical deep click feature prediction for fine-grained image recognition. IEEE Trans Pattern Anal Mach Intell. https://doi.org/10.1109/tpami.2019.2932058

36. Zhang J, Yu J, Tao D (2018) Local deep-feature alignment for unsupervised dimension reduction. IEEE Tran Image Process 27(5):2420–2432. https://doi.org/10.1109/TIP.2018.2804218

37. Zhang X, Huang S, Zhang X, Wang W, Wang Q, Yang D (2018) Residual inception: a new module combining modified residual with inception to improve network performance. In: 2018 25th IEEE international conference on image processing (ICIP). IEEE, pp 3039–3043. https://doi.org/10.1109/ICIP.2018.8451515

Publisher’s Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.