PT-CapsNet: A Novel Prediction-Tuning Capsule Network Suitable for Deeper Architectures

Chenbin Pan and Senem Velipasalar
EECS Department, Syracuse University, Syracuse, NY 13244
{cpan14, svelipas}@syr.edu *

Abstract

Capsule Networks (CapsNets) create internal representations by parsing inputs into various instances at different resolution levels via a two-phase process – part-whole transformation and hierarchical component routing. Since both of these internal phases are computationally expensive, CapsNet have not found wider use. Existing variations of CapsNets mainly focus on performance comparison with the original CapsNet, and have not outperformed CNN-based models on complex tasks. To address the limitations of the existing CapsNet structures, we propose a novel Prediction-Tuning Capsule Network (PT-CapsNet), and also introduce fully connected PT-Capsules (FC-PT-Caps) and locally connected PT-Capsules (LC-PT-Caps). Different from existing CapsNet structures, our proposed model (i) allows the use of capsules for more difficult vision tasks and provides wider applicability; and (ii) provides better than or comparable performance to CNN-based baselines on these complex tasks. In our experiments, we show robustness to affine transformations, as well as the lightweight and scalability of PT-CapsNet via constructing larger and deeper networks and performing comparisons on classification, semantic segmentation and object detection tasks. The results show consistent performance improvement and significant parameter reduction compared to various baseline models. Code is available at https://github.com/Christinepan881/PT-CapsNet.git.

1. Introduction

Convolutional Neural Networks (CNNs) [4][12][27] can capture features of objects similar to human visual system by assembling a set of small kernels looking for different patterns. Their ability to learn rich feature representations from data have allowed them to find a wide range of applications for various vision tasks.

When looking at an object, humans deconstruct it into hierarchical sub parts, and tend to develop a relationship between object parts [10]. This parsing process aligns well with the capsule networks (CapsNets) [26]. Capsules represent distinct types of parts/instances, and each capsule is a collection of neurons, encapsulating different attributes of a part/instance, e.g. scale, orientation etc. A capsule layer consists of multiple capsules. A special agreement/routing mechanism between subsequent capsule layers is used for parsing different levels of capsules. Encapsulation of different attributes and the agreement mechanism encourage each capsule to be responsible for capturing how an entity is represented, instead of only indicating its presence as in traditional neuron activations.

However, existing CapsNet variants have several limitations preventing them from being as widely adopted as CNNs. Firstly, the fully pairwise connection between capsules focuses more on global information, but is not that conducive to capturing diverse local relationships. Secondly, while stacking capsule layers, the fully pairwise connection between every capsule tends to increase the number of parameters exponentially, thereby decreasing the generalization ability and causing the overfitting problem during training. Thirdly, the routing-by-agreement mechanism [26] is computationally expensive and time consuming. Due to these limitations, current CapsNets cannot be generalized to a wider range of more complicated computer vision tasks and datasets involving deeper networks.

In order to utilize all the benefits of CapsNets and realize their full potential, we propose a novel Prediction-Tuning Capsule Network (PT-CapsNet) to overcome the limitations of previous CapsNets. We show that PT-CapsNet is a scalable and equivariant model for capsule networks, and has more sparse projections from the input capsule to output capsule space, providing better robustness and generalization ability. Different from the existing CapsNets variants, our contributions include the following: We (i) propose a novel PT-CapsNet, which is lightweight and efficient; (ii) introduce two instance layers–fully connected PT

*The information, data, or work presented herein was funded in part by National Science Foundation (NSF) under Grants 1739748 and 18167325 and by the Advanced Research Projects Agency-Energy (ARPA-E), U.S. Department of Energy, under Award Number DE-AR0000940. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

capsule layer (FC-PT-Caps) and locally connected PT capsule layer (LC-PT-Caps)– to make the PT-CapsNet applicable to various deep learning architectures; (iii) conduct ablation studies to investigate the best combination to build the PT-CapsNets; (iv) design a PT-CapsNet architecture for classification, which is composed of multiple FC-PT-Caps and LC-PT-Caps; (v) conduct experiments to validate the robustness of PT-CapsNet to affine transformations, and achieve much better results by using 29 times less number of parameters compared to previous CapsNets; (vi) scale our model to larger deep learning architectures for classification, semantic segmentation, and object detection tasks, and achieve better or comparable performance to CNN-based baselines using much less number of parameters.

2. Related Work

Considering the rich information stored in capsules, several works have been proposed to improve CapsNets. Building on the original CapsNet [26], EM-Caps [13] uses a matrix to represent the pose, and Expectation-Maximization-based routing algorithm for better performance. DeepCaps [20] aim to improve the performance of CapsNets [26] on CIFAR10 [16] dataset, and achieves 3% accuracy improvement. SOVNET [32] uses a degree-centrality routing to get an equivariant model for CapsNets. GCaps [18] is a group equivariant capsule network. Although the aforementioned CapsNets variants achieve improvement compared to original CapsNets, they can not reach a performance comparable to commonly used CNNs, and are only applicable to simple image classification tasks. The inverted dot-product attention capsule network (IDACapsNet) [29] with ResNet backbone provides 95.14% accuracy, compared to 95.11% of ResNet, on CIFAR10 by using sequential iterative routing and layer normalization. When IDACapsNet uses a simpler CNN backbone (0.56M params), it only provides 85.17% accuracy. The SR-CapsNet [11] achieves competitive performance on adversarial defense and viewpoint generalization, but still has scalability issues and higher computation cost. In [17], a local routing algorithm is proposed with an encode-decode capsule architecture for medical image segmentation. This model provides a slight improvement of 0.03% compared to U-Net [24], which is not the SOTA for semantic segmentation. In [23], attentive group convolution is introduced to highlight meaningful relationships among symmetries. SubSpace CapsNet [8] is applied on GAN, achieving SOTA performance in semi-supervised classification. In [33] and [1], the authors apply CapsNet on font style verification and COVID-19 identification, respectively. 3D CapsNets are introduced in [2],[15] for volumetric object classification, and a 3D point cloud capsule network is proposed [37] for object classification, part-segmentation and object reconstruction. Yet, these models only adopt 1-2 capsule layers as a small composition.

3. Proposed Models

In a CapsNet, each capsule is a collection of neurons, which represent different attributes of the pose of a part/instance, e.g. location, scale, orientation etc. Different ways have been employed to represent the pose, and indicate the presence probability of a part/instance. Also, a non-linearity is used in CapsNets to normalize the capsules. Sabour et al. [26] use vectors as the pose structure to represent a capsule, the l2 norm of vectors to indicate the instance presence probability, and squash function as the non-linearity. Hinton et al. [13], use a 4×4 matrix to represent a part's pose, and EM routing to determine the logistic unit and non-linearity. In our work, we propose a Prediction-Tuning Capsule Network (PT-CapsNet), which is more efficient and lightweight than the previous CapsNet structures. PT-CapsNet uses the vector-form to represent the capsule pose. To be able to handle bigger and more complex datasets, and make the proposed PT-CapsNet applicable to wider range of challenging tasks, we also propose a fully connected PT-Capsule layer in Sec. 3.2 and a locally connected PT-Capsule layer in Sec. 3.3. Then, using these two types of capsule layers, we construct a novel deep PT-CapsNet architecture for classification in Sec. 3.4.

3.1. Preliminaries

Let $X \in \mathbb{R}^{C_{in},N_{in}}$ denote the input capsule map at layer l, where C_{in} is the number of the input capsule types, and N_{in} is the dimension of the input pose vector. The function of layer l is to transfer X to higher-level capsules, $Y \in \mathbb{R}^{C_{out},N_{out}}$ at layer $l+1$. Existing CapsNet structures first apply C_{out}-many different transformation matrices, $M \in \mathbb{R}^{N_{in},N_{out}}$, to each of C_{in}-many input capsules, generating C_{in}, C_{out}-many predictions ($\vec{u} \in \mathbb{R}^{N_{out}}$) for C_{out}-many higher-level capsules. The goal is to explore the pair-wise relationships between the capsules in two consecutive layers. The produced predictions are typically called intermediate votes, which are then forwarded into the routing-by-agreement mechanism. In this phase, the part-whole relationships are further evaluated by considering the importance/contribution of the intermediate votes for each higher-level capsule. $C_{in} \times C_{out}$-many weights/coupling coefficients (w) are calculated as the agreement values between the pairs. Finally, each output capsule is produced by computing the weighted sum of the intermediate predictions made, for that output capsule, by capsules in layer l. This two-step process can be described by:

$$\vec{u}_{i,j} = \vec{x}_i \cdot M_{i,j} \tag{1}$$

$$\vec{y}_j = \sum_{i}^{C_{in}} w_{i,j} \times \vec{u}_{i,j} \ , \tag{2}$$

where $i \in [1, C_{in}]$ and $j \in [1, C_{out}]$ indicate the IDs of the i^{th} input capsule and j^{th} output capsule, respectively.

Thus, for previous CapsNets, it is necessary to train a large number of parameters for modeling the pairwise relationships between capsules in two consecutive layers. For the fully connection, a typical example is [26], wherein the total number of parameters for the transformation matrices is $C_{in} \times C_{out} \times N_{in} \times N_{out}$. Yet, after the weight distribution of the routing process, only some of the prediction vectors are emphasized, while others are compressed. The compressed vectors with less weight will not have much contribution for the higher-level capsules. This is due to the fact that, in general, not all the entities in one layer represent the components of one specific object, i.e. not all pairs of capsules have significant relationships. Hence, the pair-wise projection causes excess computation providing unnecessary or redundant information. Also, the typical dynamic routing algorithm can be easily influenced by the vectors with longer lengths regardless of whether their predictions are reliable or not. The details of this limitation are provided in the supplementary material. Thus, the large number of parameters and inefficiency of workflow inhibit stacking of many capsule layers to scale up to large architectures.

3.2. Fully Connected PT-Caps

To address the aforementioned limitations, we introduce the fully connected prediction-tuning capsule (FC-PT-Caps) layer . For the FC-PT-Caps layer, instead of having each input capsule make predictions for all output capsules, we first transform the low-level capsules, $X \in \mathbb{R}^{C_{in}, N_{in}}$, with low-level pose to the capsules with high-level pose by employing C_{in}-many transformation matrices ($M' \in \mathbb{R}^{N_{in}, N_{out}}$) to perform matrix multiplication with the corresponding input capsules. We refer to the resulting capsules with high-level pose as hidden capsules, $H \in \mathbb{R}^{C_{in}, N_{out}}$. The goal is to learn the relationships between the low-level and high-level poses of the input capsules. Then, to probe the relationships between low-level and high-level capsule types, we use vector-form weights, instead of scalar weights, to refine the hidden capsules. We refer to this process as the vector-tuning process. More specifically, when deriving each high-level capsule, there will be C_{in}-many vector-form weights ($\vec{v} \in \mathbb{R}^{N_{out}}$), which are used to perform element-wise multiplication with the corresponding C_{in}-many hidden vectors. After multiplication, the weighted hidden vectors in $\mathbb{R}^{C_{out}, C_{in}, N_{out}}$ dimensional space, are summed along the C_{in} axis to obtain the final capsules $Y' \in \mathbb{R}^{C_{out}, N_{out}}$. The figure illustrating this process is provided in the supplementary material. The overall procedure can be expressed by Eq. (3) and Eq. (4):

$$\vec{h}_i \underset{C_{in}}{=} \vec{x}_i \cdot M'_i \tag{3}$$

$$\vec{y'}_j = \sum_i \vec{v}_{i,j} \times \vec{h}_i \,, \tag{4}$$

where $i \in [1, C_{in}]$ and $j \in [1, C_{out}]$ indicate the IDs of the i^{th} input capsule and j^{th} output capsule, respectively. Hence, in the first phase, each capsule undergoes pose transformation independently, while in the second phase, each feature in the pose renders high-level capsules independently. When parsing the object relationship, if two instances are mutually exclusive, then they will not be connected, which means that there is no need for one to predict the other, and generate the paired prediction. With this in mind, we only make one prediction for each input instance. In other words, we perform advanced pose transformation on each input instance, instead of making each instance predict all high-level instances. In the second stage, the tuning mechanism will extract the required information from the hidden capsules, and fuse it to synthesize the high-level instances. In this way, we provide significant savings in memory and computational requirements, which are otherwise wasted on generating invalid pairs and redundant information. Also, the vector-form weights, used in the tuning phase, ensure that each feature in the pose of higher-level capsule is inferred from the corresponding feature in the pose of the hidden capsules, and is not impacted by other kinds of features, while in previous CapsNets, the features in each middle capsule are given the same weights when predicting outputs. In our proposed structure, the parameters in both phases are trainable, so that they can accumulate knowledge during training. In the previous CapsNets, only the first step is trainable to serve this function.

In general, different from the previous CapsNets with pair-wise transformation and routing, our FC-PT-Caps first performs capsule-wise prediction followed by feature-wise tuning. Following the above notation, the total number of training parameters for FC-PT-Caps can be calculated as follows: For the capsule-wise prediction phase, C_{in}-many transformation matrices result in $C_{in} \times N_{in} \times N_{out}$ parameters; then for the feature-wise tuning phase, $C_{in} \times C_{out}$-many vector weights lead to $C_{in} \times C_{out} \times N_{out}$-many parameters. Thus, FC-PT-Caps has a total of $C_{in} \times N_{out} \times (N_{in} + C_{out})$-many parameters compared to the $C_{in} \times C_{out} \times N_{in} \times N_{out}$-many parameters in previous CapsNets. It can be seen that the difference in the number of parameters of FC-PT-Caps and previous CapsNets is only affected by the values of N_{in} and C_{out}. In this work, we empirically choose the number of capsule-types (C) to be 2^a, where $a \in [4, 9]$, and the dimension of capsule vectors (N_{in}) is chosen from $[4, 8, 16]$. Both of these hyper-parameter choices correspond to the setup commonly used by previous CapsNets. Based on these choices, it is evident that our FC-PT-Caps is much more lightweight than others. We list the exact number of parameters for comparison in Sec 4.2 to demonstrate that our model has far less parameters than others.

While providing significant reduction in the number of parameters, our proposed model does not lose any of the functionalities of an original CapsNet, and can still parse

the hierarchical component structure of visual objects. To prove this, let us consider the same input X, and denote the outputs from previous CapsNets and our FC-PT-Caps by Y and Y', respectively. If both outputs can be derived from the same composition structure based on X, we can say that it is possible for the previous CapsNets and FC-PT-Caps to explore an equivalent feature space. This definition preserves the transformation from input to output space, meaning that the information in the hierarchical composition structure will not be lost.

For previous CapsNets, when Eq. (1) and Eq. (2) are combined, we obtain $\vec{y}_j$ as shown in Eq. (5):

$$\vec{y}_j = \sum_{i}^{C_{in}} w_{i,j} \times (\vec{x}_i \cdot M_{i,j}). \tag{5}$$

Based on the commutative property and associative property, the terms of summation above can be written as:

$$w_{i,j} \times (\vec{x}_i \cdot M_{i,j}) = \vec{x}_i \cdot (w_{i,j} \times M_{i,j}). \tag{6}$$

We can set $T_{i,j} = w_{i,j} \times M_{i,j}$, where the matrix $T \in \mathbb{R}^{N_{in},N_{out}}$ has two components coming from routing-by-agreement and voting procedure, respectively. Then, Eq. (5) can be rewritten as follows:

$$\vec{y_j} = \sum_{i}^{C_{in}} \vec{x}_i \cdot T_{i,j}. \tag{7}$$

Similarly, for our FC-PT-Caps, when Eq. (3) and Eq. (4) are combined, $\vec{y'}_j$ can be written as in Eq. (8):

$$\vec{y'}_j = \sum_{i}^{C_{in}} \vec{v}_{i,j} \times (\vec{x}_i \cdot M'_i). \tag{8}$$

We can then write

$$\vec{v}_{i,j} \times (\vec{x}_i \cdot M'_i) = \vec{x}_i \cdot (V'_{i,j} \times M'_i), \tag{9}$$

where

$$V'_{i,j} = tile(extend(\vec{v}_{i,j})), \tag{10}$$

i.e., $V' \in \mathbb{R}^{N_{in},N_{out}}$ is obtained by first extending $\vec{v} \in \mathbb{R}^{N_{out}}$ into matrix $V \in \mathbb{R}^{1,N_{out}}$, and then tiling V along the first axis by N_{in} times. Then, we use matrix $T' \in \mathbb{R}^{N_{in},N_{out}}$ such that $T'_{i,j} = V'_{i,j} \times M'_i$. It is apparent that T' is composed of parameters from the prediction phase and the vector-tuning phase. Thus, Eq. (8) can be rewritten as:

$$\vec{y'}_j = \sum_{i}^{C_{in}} \vec{x}_i \cdot T'_{i,j}. \tag{11}$$

Comparing Eq. (7) and (11), it is clear that both $\vec{y}$ and $\vec{y'}$ are constructed by projecting input capsules $\vec{x}$ into $\mathbb{R}^{N_{out}}$ space, via matrices in $\mathbb{R}^{N_{in},N_{out}}$ dimensional space, and then summing the transferred vectors. Thus, we can reach the conclusion that although the processes are different, $\vec{y}$ and $\vec{y'}$ can be projected into similar feature space through previous CapsNets and FC-PT-Caps, respectively. In addition, our FC-PT-Caps can reach the same destination using less parameters, which means that our projection from input to output space is more sparse. Hence, the overfitting problem, which usually affects the previous CapsNets, can be properly avoided, making our model more flexible to be generalized to more complex datasets. We also validate the robustness of our model in Sec. 4.2.

3.3. Locally Connected PT-Caps

Fully connected (FC) capsule layers focus more on extracting global information, but are not that conducive to capturing diverse local relationships between adjacent locations, which are very important for many computer vision tasks. Stacking only the FC layers for CapsNet will also generate a large amount of parameters, requiring extensive memory and computational resources, and leading to weak generalization ability and the overfitting problem. Hence, to address these issues, and further enhance the applicability of our model, we also propose a locally connected PT-CapsNet, which is referred to as the LC-PT-Caps layer.

Instead of one capsule-type corresponding to a single capsule like in an FC layer, in locally connected layer, one capsule-type encloses a map of capsules. Therefore, to represent the flow between different LC-PT-Caps layers, the capsule tensor domain also contains location axes, in addition to the capsule-type axis and capsule dimension axis. Let $X^{LC} \in \mathbb{R}^{C_{in},N_{in},H_{in},W_{in}}$ and $Y^{LC} \in \mathbb{R}^{C_{out},N_{out},H_{out},W_{out}}$ denote the input and output feature maps, respectively, for the LC-PT-Caps layer l. Similar to the FC-PT-Caps layer, we first evolve the low-level pose of an input capsule map to high-level pose. For each type of capsule map, we use a sliding window of matrices in $\mathbb{R}^{N_{in},N_{out}}$, with the reception field of $[K_1, K_1]$ shared among different locations, to do the matrix multiplication with each capsule vector within the reception field. The resulting vectors in one field are summed to get the hidden capsule vector at the corresponding location and capsule-type. Concatenating the hidden vectors based on capsule-type, we get the hidden map $H^{LC} \in \mathbb{R}^{C_{in},N_{out},H_{hid},W_{hid}}$. This process can be expressed as:

$$\vec{h}^{LC}_{i,a,b} = \sum_{p}^{K_1} \sum_{q}^{K_1} \vec{x}^{LC}_{i,a_p,b_q} \cdot M^{LC}_{i,p,q}, \tag{12}$$

where $\vec{h}^{LC}_{i,a,b}$ and $\vec{x}^{LC}_{i,a_p,b_q}$ indicate the evolved hidden vector at $[a, b]$ position, and the capsule vector at $[a_p, b_q]$ position of i^{th} input capsule-type, respectively, and $M^{LC}_{i,p,q}$ denotes the matrix at $[p, q]$ position of the sliding window for the i^{th} input capsule-type. This operation is capsule-type-wise, so with the shared matrices, one type of input capsules will make predictions for one type of hidden capsules. Then, there will be C_{in}-many groups of sliding matrices, which

generate $C_{in} \times K_1 \times K_1 \times N_{in} \times N_{out}$-many parameters for the first prediction stage.

For the second stage of LC-PT-Caps, similar to the tuning step in FC-PT-Caps, we adjust and fuse information from hidden capsules to get the higher-level capsules. To produce each output capsule map in $\mathbb{R}^{N_{out},H_{out},W_{out}}$, we use a sliding cube of weights, $V^{LC} \in \mathbb{R}^{K_2,K_2,N_{out}}$, to perform element-wise multiplication at positions of each type of hidden capsule map, where $[K_2, K_2]$ represents the window size (reception field size), and N_{out} represents the weights of the N_{out} features at each position. Then, to exploit the features from each capsule type, the weighted hidden capsule vectors are summed together to get the high-level capsule vector at the corresponding location. By concatenating the resulting capsule maps based on output capsule-type, we can get the final map $Y^{LC} \in [C_{out}, N_{out}, H_{out}, W_{out}]$. This process can be expressed as:

$$\vec{y}^{LC}_{j,a,b} = \sum_{i}^{Cin} \sum_{p}^{K_2} \sum_{q}^{K_2} \vec{v}^{LC}_{j,i,p,q} \times \vec{h}^{LC}_{i,a_p,b_q}, \tag{13}$$

where $\vec{v}^{LC}_{j,i,p,q}$ is the vector weights at $[p, q]$ position of the sliding cube for i^{th} hidden capsule map to produce j^{th} output capsule map, $\vec{y}_{j,a,b}$ is the capsule vector at $[a, b]$ position of the j^{th} output capsule map, and $\vec{h}^{LC}_{i,a_p,b_q}$ is the capsule vector at $[a_p, b_q]$ position of the i^{th} hidden capsule group. The second stage is feature-wise adjustment, such that for predicting one type of capsule map, C_{in}-many shared cubes will be required, which leads to $C_{out} \times C_{in} \times K_2 \times K_2 \times N_{out}$-many parameters. Therefore, the total number of parameters for LC-PT-Caps layer is $(C_{in} \times K_1 \times K_1 \times N_{in} \times N_{out}) + (C_{out} \times C_{in} \times K_2 \times K_2 \times N_{out}) = C_{in} \times N_{out} \times (K_1 \times K_1 \times N_{in} + K_2 \times K_2 \times C_{out})$. The figure illustrating this process is provided in the supplementary material. The procedure in LC-PT-Caps is similar to that in FC-PT-Caps, which is also a prediction-tuning process. For both phases in the LC-PT-Caps layer, K determines the reception field size when capturing local features, and due to the shared weights among locations, it is also a lightweight structure compared to previous CapsNets.

3.4. PT-CapsNet for Classification

We now propose a novel PT-CapsNet architecture for classification by using our FC-PT-Caps and LC-PT-Caps layers. The model shown in Fig. 1 is composed of six main blocks: one convolution block, four LC-PT-Caps blocks, and one FC-PT-Caps block. The convolution block is used to extract the initial features from the input images. It contains a 3×3 convolution layer, followed by a batch normalization (BN) layer and a ReLU activation layer. To transfer the initial features to capsule domain, we add an additional axis to the feature map, representing the capsule vector dimension, so that the initial capsule vector dimension is 1.

For each LC-PT-Caps block, there are five capsule units and one concatenation unit as shown in Fig. 1. For each capsule unit, we adopt BN right after one LC-PT-Caps layer and before the non-linearity function. We set $K_1 = 1$ and $K_2 \in [1, 3]$ for all the LC-PT-Caps layers. The first capsule unit in each block (purple squares) is treated as a transition unit to process the input capsule maps from the previous block. The transition unit in the first LC-PT-Caps block has $K_2 = 3$, while the rest has $K_2 = 1$. The second capsule unit is used to change the size of the capsule feature maps by modifying the stride of the sliding cube at the second phase, hence it is referred to as the down-sampling unit (blue squares). We do not change the feature map size in the first LC-PT-Caps block, where the stride is set to be 1. For the remaining three blocks, we set the stride to be 2. The third and fourth capsule units (pink squares) are used to further process the capsule outputs from the down-sampling block. We set K_2 to 3 and 1 for these blocks, and stride equal to 1. The down-sampling unit and the following two units together form a sequential structure to study a mapping for the outputs from the transition unit. The fifth unit, referred to as the inception unit (green squares), is used to learn a different mapping for the outputs from the transition unit. K_2 is set to be 1, and to match the feature map size for this connection, we set the stride equal to the stride in the parallel down-sampling unit. The concatenation unit is used to merge the outputs from the sequential block and the inception unit along the capsule-type axis. This architecture is a combination of the two mappings with their outputs concatenated into a single capsule output domain. In this way, we scale the CapsNet width by widening the capsule-type channel for the feature maps to make the model capture various instances and easier to train.

After four LC-PT-Caps blocks, we have the FC-PT-Caps block as the final classification block. For the feature map generated from the last LC-PT-Caps block, we concatenate the H, W axes with the capsule-type axis to reshape it into the FC capsule domain, which only has the capsule-type and feature-dimension axes. The resulting feature map is in $\mathbb{R}^{H \times W \times C,N}$. Then, we adopt one FC-PT-Caps layer followed by BN and a non-linearity function to project the feature map into the class space $\mathbb{R}^{cls,16}$, where cls represents the number of class. To find the best way of acquiring the final logits of the class capsules, we conduct an ablation study to compare the typical l2 norm logits and 'generated logits', which refer to using an additional FC-PT-Caps layer to generate capsules with only one element representing the classification probability. Since our experimental results show that the 'generated logits' perform better than the l2 norm logits, we add another FC-PT-Caps layer, in which the output capsule domain is $\mathbb{R}^{[cls,1]}$, to get the final prediction. Visualization of focus of capsules in the 2nd transition unit are provided in Fig. 2 to illustrate the semantic information

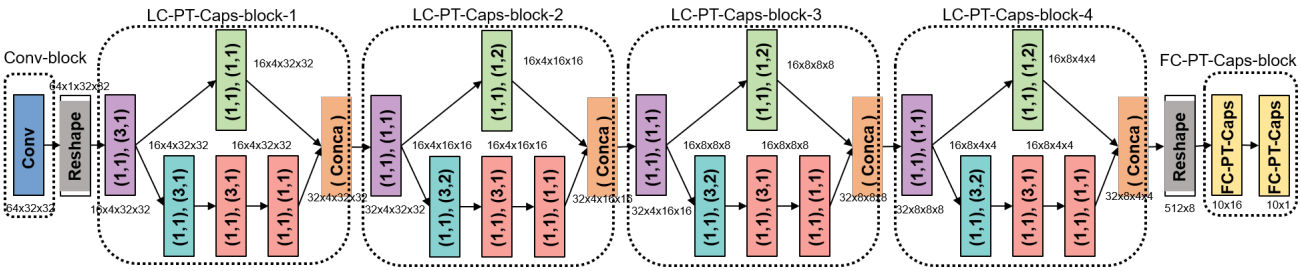

Figure 1. A novel PT-CapsNet constructed for image classification. Purple and blue squares represent the transition unit and the down-sampling unit, respectively, pink squares are the units further processing the capsules, and green square represents the inception unit. The LC-PT-Caps units are described by $(K_1, stride1), (K_2, stride2)$ in the figure.

represented by each level of capsules.

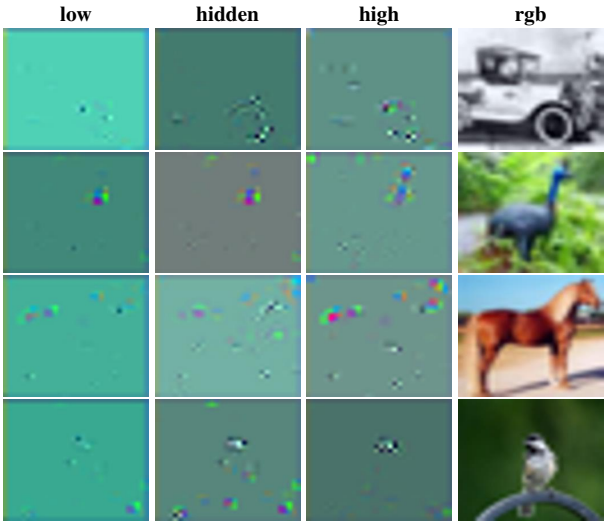

Figure 2. Visualization of different levels of capsules.

4. Experiments

Our experiments have three parts: (i) to investigate the effects of different factors on the model performance and show the effectiveness of our model, we first conduct an ablation study; (ii) to demonstrate the robustness of PT-CapsNet, we compare its performance with six different previous CapsNet models under affine transformations on CIFAR10 [16]; (iii) to show the lightweight, scalability and wider applicability of PT-CapsNet, we scale it up to larger deep learning architectures for *different tasks*, namely classification on CIFAR10, CIFAR100 [16], and FashionMNIST datasets, semantic segmentation on ISIC2018 [6][30] dataset, and object detection on PASCAL VOC dataset [9].

4.1. Ablation Study

Our ablation study focuses on three aspects of the PT-CapsNet. First, we investigate on the prediction and tuning type/order. For the PT-CapsNet introduced in Sec. 3, the capsule-wise prediction for high-level pose is performed first, followed by the feature-wise tuning for the higher-level capsule-type. Yet, it is also reasonable to first perform feature-wise prediction for higher-level capsule-type, followed by the capsule-wise tuning for higher-level pose. The second aspect of ablation study is about the logits. l2 norm is used most commonly to calculate the logits for vector-form capsules [26]. We argue that it is also sensible to apply an additional capsule layer to generate class logits for each capsule. The third aspect of the ablation study is related to the non-linearity functions. In original CapsNet [26], the squash function is used to normalize capsules. In [3], ReLU function is employed as the non-linearity for the primary capsules. The swish function [21] is also proved to work well with large architectures and advanced tasks [28].

To investigate the best prediction and tuning type/order, the logit generation method, and the non-linearity function, we conduct experiments on MNIST [7], FashionMNIST [35], and KMNIST [5] datasets covering 12 combinations shown in Tab. 1. We use a simpler architecture for this part, which is composed of two convolution layers as backbone, and two PT-Capsule layers—one LC-PT-Caps and one FC-PT-Caps layer—as the capsule part. The convolution layers are followed by BN and ReLU activation, and the capsule layers are followed by BN and the choice of non-linearity. The number of channels for the two convolution layers are 64 and 128, and the kernel size and stride for both layers are 3×3 and 2, respectively. The number of capsule types and capsule dimension for the LC-PT-Caps and FC-PT-Caps are $[32, 8]$ and $[10, 16]$, respectively. The LC-PT-Caps has a reception field of 3×3 with cubic stride of 2. Each model is trained for 100 epochs with SGD optimizer, and the batch size and initial learning rate are equal to 128 and 0.1, respectively. The learning rate decay is 0.1 for every 50 epochs. We perform 4-pixel zero padding at all sides, and do horizontal flip with a probability of 0.5 for data augmentation. The testing error in Tab. 1 is calculated as the average value of 5 runs. It can be seen from Tab. 1 that (i) models with capsule-wise prediction for high-level pose and the feature-wise tuning for the higher-level capsule-type perform better than the models with feature-wise prediction for higher-level capsule-type and the capsule-wise tuning for higher-level pose in most cases; (ii) 'Generated logits' (GL) consistently outperforms the l2-norm based logits

method; and (iii) although the squash function works well with l2-norm based method, it cannot surpass the performance of ReLU function used with the GL method. Thus, to construct the best PT-CapsNet, we have combined the most promising options—pose prediction, GL method, and ReLU–for the remainder of the experiments.

Models			Testing error		
Prediction	Logits	Activation	MNIST	F-MNIST	KMNIST
Feature-wise prediction	GL	ReLU	0.75	8.29	4.35
		Squash	0.81	8.42	5.13
		Swish	0.76	8.12	4.2
	L2	ReLU	88.65	90	90
		Squash	1.66	10.68	10.39
		Swish	42.26	21.76	10.36
Capsule-wise prediction	GL	ReLU	**0.63**	**7.77**	**3.47**
		Squash	0.74	7.84	5.02
		Swish	0.64	8.1	3.58
	L2	ReLU	88.65	90	90
		Squash	1.52	10.44	9.54
		Swish	89.9	74.4	66.06

Table 1. Ablation study on MNIST, Fashion-MNIST, and KMNIST datasets.

4.2. Robustness to Affine Transformations

We compare our classification model, which has 0.29M parameters and is described in Sec. 3.4, with six CapsNet-based methods with respect to robustness to affine transformations for classification on CIFAR10. The methods we compare with and their number of training parameters are CapsNet (8.5M) [26], EM-Caps (0.32M) [13], G-Caps (7.8M) [18], SR-CapsNet (3.2M) [11], DeepCaps (8.5M) [20], and SOVNET (7.3M) [32].

Our affine transformation methods are the same as in [32]. We created five variations of the training and test sets by randomly transforming data. We consider the combination of translation and rotation. We choose the translation extent and the rotation degree from the following 5 combinations: [0 pixel, 0°], [2 pix., 30°], [2 pix., 60°], [2 pix., 90°], and [2 pix., 180°]. We train each model on the five transformed versions of the training set separately, and then test them on all five transformed versions of the test set. Thus, for each model, there are 25 robustness evaluation results summarized in Table 2. Our PT-CapsNet outperforms others in most of the cases, indicating that PT-CapsNet is more robust. In only eight out of 25 cases, SR-CapsNet, SOVNET, and DeepCaps provide better accuracy, but their number of parameters are much higher than ours (by almost 11, 25, and 29 times). Furthermore, the performance of PT-CapsNet on the un-transformed dataset is much better than the other CapsNet baselines. PT-CapsNet having the least number of parameters among others further demonstrates that the robustness mostly comes from the framework and the sparse projection space.

Training on Untransformed CIFAR-10					
Models	(0,0°)	(2,30°)	(2,60°)	(2,90°)	(2,180°)
CN[26]	68.28	55.57	43.55	37.48	30.89
EM[13]	62.85	49.28	41.27	34.73	29.9
GC[18]	49.54	38.45	31.89	30.88	27.7
SR[11]	**91.49**	62.25	42.18	36.89	30.26
DC[20]	76.76	67.97	**53.56**	45.22	35.67
SN[32]	88.34	47.57	42.24	43.75	**43.52**
Ours	91.21	**70.18**	50.41	**46.79**	40.56
Training on Affine Transformation by (2,30°)					
Models	(0,0°)	(2,30°)	(2,60°)	(2,90°)	(2,180°)
CN[26]	73.45	69.87	61.17	52.29	45.58
EM[13]	70.24	66.63	59.10	50.93	42.26
GC[18]	49.5	48.88	45.75	42.93	38.74
SR[11]	**90.36**	**89.15**	78.07	65.37	50.07
DC[20]	84.24	82.54	74.63	63.54	48.63
SN[32]	86.58	85.35	82.51	**79.14**	69.64
Ours	90.16	88.41	**85.2**	78.5	**70.7**
Training on Affine Transformation by (2,60°)					
Models	(0,0°)	(2,30°)	(2,60°)	(2,90°)	(2,180°)
CN[26]	70.26	67.69	66.62	60.04	47.99
EM[13]	66.53	65.09	63.21	58.04	47.61
GC[18]	49.63	50.31	48.84	47.43	43.11
SR[11]	86.18	**87.78**	84.68	80.21	60
DC[20]	83.92	83.63	82.79	78.09	60.02
SN[32]	82.86	83.63	83.57	83.06	80.89
Ours	**87.34**	86.82	**85.49**	**84.97**	**81.7**
Training on Affine Transformation by (2,90°)					
Models	(0,0°)	(2,30°)	(2,60°)	(2,90°)	(2,180°)
CN[26]	67.81	65.64	65.46	64.35	52.79
EM[13]	64.33	63.0	62.7	61.42	52.08
GC[18]	49.98	51.24	50.63	49.95	46.59
SR[11]	85.17	83.97	83.26	82.15	67.73
DC[20]	82.91	82.78	82.66	82.62	68.34
SN[32]	83.33	82.76	82.58	82.79	**82.22**
Ours	**85.42**	**84.77**	**83.85**	**82.84**	75.91
Training on Affine Transformation by (2,180°)					
Models	(0,0°)	(2,30°)	(2,60°)	(2,90°)	(2,180°)
CN[26]	61.08	59.53	60.04	59.85	59.9
EM[13]	57.57	55.89	56.85	56.35	55.2
GC[18]	39.09	41.03	41.43	41.25	41.08
SR[11]	82.32	81.17	81.02	80.88	80.35
DC[20]	81.12	80.81	80.64	81.05	80.92
SN[32]	82.50	81.80	81.78	81.95	81.82
Ours	**82.76**	**82.72**	**82.68**	**82.02**	**82.51**

Table 2. Affine transformation experiments on CIFAR-10 dataset.

4.3. Comparison with CNN-based Networks on Advanced Vision Tasks

We conduct image classification, semantic segmentation, and object detection experiments to compare the proposed PT-CapsNets with several CNN-based models on various datasets. To perform a commensurate comparison, we reproduced all the models in PyTorch framework [19], and report our reproduced results. For image classification, we use CIFAR-10 [16], CIFAR-100 [16], and

Models	top-1 acc.	MAC.	# params
ResNet	92.69	255.27M	1.73M
PT-CapsResNet	**93.59**	**225.7M**	**1.12M**
WRN	95.95	5.25G	36.48M
PT-CapsWRN	**95.98**	**3.51G**	**19.21M**
DenseNet	95.33	296.48M	769.16K
PT-CapsDenseNet	**95.71**	**262.38M**	**727.11K**

Table 3. Image classification results on CIFAR10 dataset.

Models	top-1 acc.	MAC.	# params
ResNet	70.32	255.28M	1.74M
PT-CapsResNet	**71.39**	**225.79M**	**1.16M**
WRN	81.09	5.25G	36.54M
PT-CapsWRN	**81.58**	**3.51G**	**19.23M**
DenseNet	77.12	296.51M	800.03K
PT-CapsDenseNet	**78.36**	**261.03M**	**655.28K**

Table 4. Image classification results on CIFAR100 dataset.

Models	top-1 acc.	MAC.	# params
ResNet	94.36	195.22M	1.73M
PT-CapsResNet	**95.28**	**171.5M**	**1.12M**
WRN	95.34	4.02G	36.48M
PT-CapsWRN	**95.57**	**2.68G**	**19.21M**
DenseNet	95.39	226.66M	768.73K
PT-CapsDenseNet	**95.99**	**200.37M**	**726.45K**

Table 5. Image classification results on FashionMNIST dataset.

Fashion-MNIST [35] datasets, and adopt ResNet-110 [12], WRN-28-10 [36], and DenseNet-100 [14] as our baseline models. For semantic segmentation, we use ISIC2018 dataset [6][30], and adopt U-Net [24] and DeepLabv3+ [4] as the baselines. For object detection, we use PASCAL VOC dataset [9], and compare with YOLO-v5 [31, 22].

We replace the convolution layer and the FC layer in the baselines with LC-PT-Caps layer and FC-PT-Caps layer, respectively, to build our PT-CapsNet-based models. For PT-DeepLabv3+ and PT-YOLO-v5, we adopt ResNet-101 pretrained on ImageNet [25] and CSPNet [34], as the backbones, respectively. For the capsule network, both the number of capsule types and the capsule dimension need to be considered. The hyperparameters used for the PT-CapsNets are provided in the supplementary material together with all the implementation details and some visualization results.

The results for image classification are summarized in Tables 3, 4 and 5 for three different datasets. We report the top-1 accuracy, to validate the effectiveness, and the total number of parameters (#params) and the total number of multiply-and-accumulates (MAC.) to show the compactness of our PT-CapsNets. It can be seen that, the PT-CapsNet slightly outperforms all three of the CNN baselines, while providing considerable reduction in the number of parameters and the amount of computation at the same time. The segmentation results are summarized in Table 6, listing the mean Intersection over Union (mIoU) and the number of parameters for each model. PT-CapsNets can

Models	mIoU	# params
Unet	65.57	31.1M
PT-Caps-Unet	**66.86**	**19.8M**
DeepLabv3+	82.85	59.3M
PT-Caps-DeepLabv3+	**83.12**	**44.6M**

Table 6. Semantic segmentation results on the ISIC2018 dataset.

Models	mAP1	mAP2	Prec.	Recall	# param
YOLO-v5	78	**52.21**	54.28	**83.27**	7.3M
PT-Caps	**78.2**	52	**61.2**	81	**6.3M**

Table 7. Object detection results on PASCAL VOC dataset

not only improve the mIoU of UNet and DeepLabv3+ by 1.31% and 0.27%, but also provide 36% and 25% parameter reduction, respectively. For object detection, mean average precision (mAP) is used as the performance metric, and the results are evaluated for mAP @.5 (mAP1), mAP @ [.5:.95] (mAP2), precision, and recall in Table 7. We can see that PT-CapsNet based detection model provides higher precision and comparable mAP with respect to YOLO-v5, with less number of parameters.

Across all different tasks and comparison experiments, the PT-CapsNet not only achieves better or on-par performance compared to baselines, but also provides significant parameter reduction, indicating that the performance enhancement and robustness is mostly due to the effective feature descriptor and proposed PT-capsule structure. This shows the great potential for PT-CapsNet to be adopted in wider range of applications. Example output images are presented in the supplementary material.

5. Conclusion and Future Work

We have presented a novel capsule network structure with prediction-tuning mechanism (PT-CapsNet) to utilize the rich information capacity of capsule networks, and address their limitations. To make the PT-CapsNet widely applicable, we have introduced fully and locally connected PT-Capsule layers, and used them to build a PT-CapsNet architecture for classification. We have compared its robustness to affine transformations with several CapsNets baselines. To show the scalability of PT-CapsNet, we have built large deep learning architectures for classification, segmentation, and object detection tasks, and compared the performance with CNN-based baselines. This is among the first works demonstrating a capsule network-based architecture can outperform or achieve on-par performance to the CNN-based models on various tasks with challenging datasets and larger image sizes, and simultaneously reduce the number of network parameters. The promising results combined with significant parameter reduction indicate that improvements are due to the proposed effective structure. Since we focused on the framework and structure, the research related to the choice of receptor field size will be future work.

References

[1] Parnian Afshar, Shahin Heidarian, Farnoosh Naderkhani, Anastasia Oikonomou, Konstantinos N Plataniotis, and Arash Mohammadi. Covid-caps: A capsule network-based framework for identification of covid-19 cases from x-ray images. *Pattern Recognition Letters*, 138:638–643, 2020. 2

[2] A. Ahmad, B. Kakillioglu, and S. Velipasalar. 3d capsule networks for object classification from 3d model data. In *2018 52nd Asilomar Conference on Signals, Systems, and Computers*, pages 2225–2229, 2018. 2

[3] John Brandt. Spatio-temporal crop classification of low-resolution satellite imagery with capsule layers and distributed attention. *arXiv preprint arXiv:1904.10130*, 2019. 6

[4] Liang-Chieh Chen, George Papandreou, Florian Schroff, and Hartwig Adam. Rethinking atrous convolution for semantic image segmentation. *arXiv preprint arXiv:1706.05587*, 2017. 1, 8

[5] Tarin Clanuwat, Mikel Bober-Irizar, Asanobu Kitamoto, Alex Lamb, Kazuaki Yamamoto, and David Ha. Deep learning for classical japanese literature. *arXiv preprint arXiv:1812.01718*, 2018. 6

[6] Noel Codella, Veronica Rotemberg, Philipp Tschandl, M Emre Celebi, Stephen Dusza, David Gutman, Brian Helba, Aadi Kalloo, Konstantinos Liopyris, Michael Marchetti, et al. Skin lesion analysis toward melanoma detection 2018: A challenge hosted by the international skin imaging collaboration (isic). *arXiv preprint arXiv:1902.03368*, 2019. 6, 8

[7] Li Deng. The mnist database of handwritten digit images for machine learning research [best of the web]. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012. 6

[8] Marzieh Edraki, Nazanin Rahnavard, and Mubarak Shah. Subspace capsule network. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 10745–10753, 2020. 2

[9] Mark Everingham, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. The pascal visual object classes (voc) challenge. *International journal of computer vision*, 88(2):303–338, 2010. 6, 8

[10] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587, 2014. 1

[11] Taeyoung Hahn, Myeongjang Pyeon, and Gunhee Kim. Self-routing capsule networks. *Advances in neural information processing systems*, 32:7658–7667, 2019. 2, 7

[12] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016. 1, 8

[13] Geoffrey E Hinton, Sara Sabour, and Nicholas Frosst. Matrix capsules with em routing. In *International conference on learning representations*, 2018. 2, 7

[14] Simon Jégou, Michal Drozdzal, David Vazquez, Adriana Romero, and Yoshua Bengio. The one hundred layers tiramisu: Fully convolutional densenets for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, pages 11–19, 2017. 8

[15] B. Kakillioglu, A. Ren, Y. Wang, and S. Velipasalar. 3d capsule networks for object classification with weight pruning. *IEEE Access*, 8:27393–27405, 2020. 2

[16] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009. 2, 6, 7

[17] Rodney LaLonde and Ulas Bagci. Capsules for object segmentation. *arXiv preprint arXiv:1804.04241*, 2018. 2

[18] Jan Eric Lenssen, Matthias Fey, and Pascal Libuschewski. Group equivariant capsule networks. *arXiv preprint arXiv:1806.05086*, 2018. 2, 7

[19] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. In *Advances in neural information processing systems*, pages 8026–8037, 2019. 7

[20] Jathushan Rajasegaran, Vinoj Jayasundara, Sandaru Jayasekara, Hirunima Jayasekara, Suranga Seneviratne, and Ranga Rodrigo. Deepcaps: Going deeper with capsule networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10725–10733, 2019. 2, 7

[21] Prajit Ramachandran, Barret Zoph, and Quoc V Le. Searching for activation functions. *arXiv preprint arXiv:1710.05941*, 2017. 6

[22] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016. 8

[23] David Romero, Erik Bekkers, Jakub Tomczak, and Mark Hoogendoorn. Attentive group equivariant convolutional networks. In *International Conference on Machine Learning*, pages 8188–8199. PMLR, 2020. 2

[24] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015. 2, 8

[25] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015. 8

[26] Sara Sabour, Nicholas Frosst, and Geoffrey E Hinton. Dynamic routing between capsules. *arXiv preprint arXiv:1710.09829*, 2017. 1, 2, 3, 6, 7

[27] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014. 1

[28] Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International Conference on Machine Learning*, pages 6105–6114. PMLR, 2019. 6

[29] Yao-Hung Hubert Tsai, Nitish Srivastava, Hanlin Goh, and Ruslan Salakhutdinov. Capsules with inverted dot-product attention routing. *arXiv preprint arXiv:2002.04764*, 2020. 2

[30] Philipp Tschandl, Cliff Rosendahl, and Harald Kittler. The ham10000 dataset, a large collection of multi-source dermatoscopic images of common pigmented skin lesions. *Scientific data*, 5:180161, 2018. 6, 8

[31] ultralytics. Yolo v5 on github. https://github.com/ultralytics/yolov5.git, 2020. 8

[32] Sairaam Venkatraman, S Balasubramanian, and R Raghunatha Sarma. Building deep, equivariant capsule networks. *arXiv preprint arXiv:1908.01300*, 2019. 2, 7

[33] T Vijayakumar and R Vinothkanna. Capsule network on font style classification. *Journal of Artificial Intelligence*, 2(02):64–76, 2020. 2

[34] Chien-Yao Wang, Hong-Yuan Mark Liao, Yueh-Hua Wu, Ping-Yang Chen, Jun-Wei Hsieh, and I-Hau Yeh. Cspnet: A new backbone that can enhance learning capability of cnn. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pages 390–391, 2020. 8

[35] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017. 6, 8

[36] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. *arXiv preprint arXiv:1605.07146*, 2016. 8

[37] Yongheng Zhao, Tolga Birdal, Haowen Deng, and Federico Tombari. 3d point capsule networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019. 2