ResCap: Residual Capsules Network for Medical Image Segmentation

Chanh D.Tr. Nguyen[1*], Huu-Hung Dao[2] and Minh-Thanh Huynh[1]

[1] FWI, FPT Software, Lot T2, D1 Street, Saigon Hi-Tech Park, Tan Phu Ward, District 9, Ho Chi Minh City, Vietnam

[2] MFG, FPT Japan, CROSS PLACE Hamamatsucho 6F, Shibakoen 1-7-6, Minatoku, Tokyo 105-001, Japan

[*]chanhndt1@fsoft.com.vn

Abstract. Convolutional neural networks (CNNs) have shown remarkable results for a wide range of task in computer vision. However, CNNs has the limitation of poor translation invariance and lack of information about pose; thus, it requires a lot of data. Capsule networks, however, have the ability to preserve information about the pose. In this paper, we present a capsule-based network for medical image segmentation. We adopt the contracting path of the U-Net architecture. The network achieves the same accuracy as U-Net but is much smaller (0.16% number of parameters compared with U-Net).

Keywords: Capsule Network, U-Net, ResNet.

1 Dynamic routing capsule

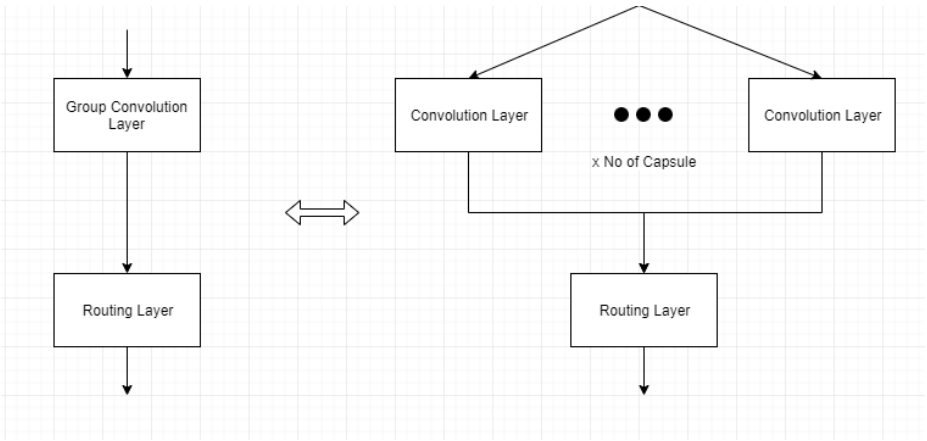

Fig. 1. Equivalent perspective of capsule. (**a**) Group convolution layer follows by a routing layer. (**b**) Convolution layer for each capsule type follows by a routing layer

In [1], S. Sabour et al. proposed a new network architecture called capsule that captures part-whole relationships of objects in an image. The capsule calculation procedure contains 2 main steps (as shown in Fig. 1): (i) *Linear transformation:* linear transform part feature so that they can be combined together to form more abstract objects with the combined feature. (ii) *Routing Procedure*: fine-tune connection between lower-level capsule with a higher one, since not all part belongs to the same object, fine-tuning connection helps assign which parts belong to which objects. We can treat these 2 steps as a sub-network with 2 layers: Group Convolution, Routing Layer.
All the convolutional layer has no activation function

$$u = Wx \tag{1}$$

For routing layer, all-paths get weighted summed after determining suitable coefficients

$$s = \sum_i \alpha_i u_i \tag{2}$$

All coefficients are calculated by routing procedure that iteratively refines them. The initial coefficients are all equal, they're then get refined by measuring how much each capsule contributes to the final result i.e. the scalar product between s and u_i.

2 Non-linear transformation in capsule

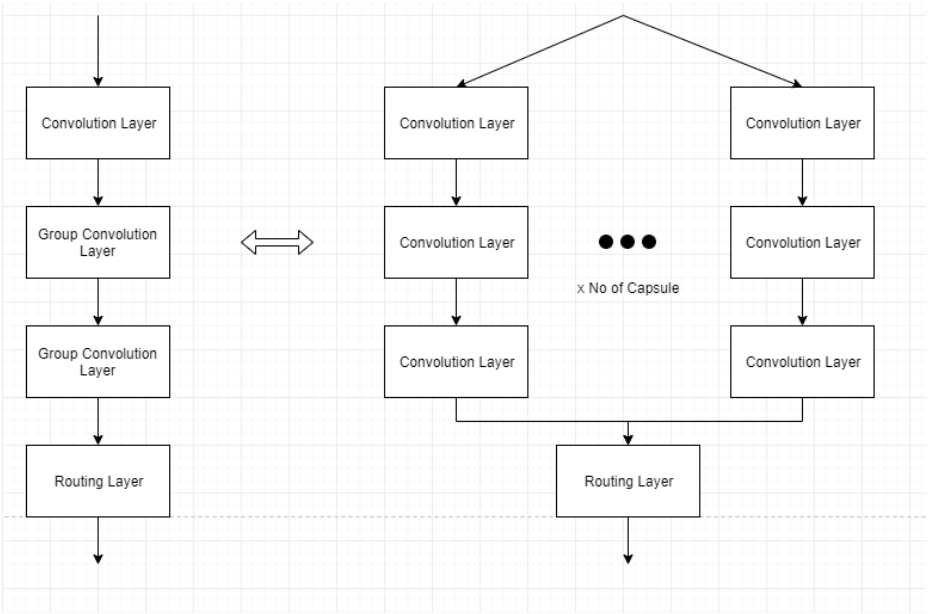

Fig. 2. Equivalent Non-linear transform capsule. (**a**) ResNext block follows by a routing layer. (**b**) Equivalent ResNext block treated as many convolutional layers

The first layer of a capsule subnetwork can be understood as affine transformation of instantiation parameters which work well for affine transformed data. However, real

world data transformation is non-linear e.g. transformation in medical image is non-homogenous.
Inspired by [2], we replace the group convolution layer with a sub-network with 3 layers to make it a non-linear transformation as shown in ***Fig.*** *2*:

$$u = f(x) \tag{3}$$

For the final group convolution layer, we don't use activation function as we want to use capsule as a residual part of the network (see Section 3.) just like in [3]

3 Residual capsule block

Normal convolution layer is already a good pattern extractor, so we treat any additional hierarchical instantiation feature as augmented information which helps strengthen the effect of original convoluted feature. In [2], the information is aggregated by summation over each residual path. This summation can be treated as normal neural network inner product with each weights equal unity (the so called "Network-in-Neuron").

$$h = x + \sum_{i=1}^{C} T_i(x) \tag{4}$$

In order to reflect hierarchical relationship aggregation in image using (3), we add routing coefficient for each $T_i(x)$ and treat each of them as a part extractor i.e. a capsule type

$$h = x + \sum_{i=1}^{C} \alpha_i T_i(x) \tag{5}$$

Each α_i is routing coefficient expressing how much each component of $T_i(x)$ can interact with each other to form an object. Here we treat each capsule as a convolutional network with linear last layer, then we fine-tune the contribution of each capsule by using Dynamic Routing procedure. Since we only have 1 output capsule, we modify the routing algorithm as followed: (**i**) we change softmax to sigmoid as each capsule is independent from one another. (**ii**) we don't use squash function as that would diminish the effect of residual sum by limiting the value in $[-1, 1]$.

---

procedure $Routing(T_i(x), t)$

$for\ all\ path\ i\ in\ the\ block: \beta_i \leftarrow 0$
  for $t\ iterations$ ***do***
    $for\ all\ path\ i\ in\ the\ block: \alpha_i = sigmoid(\beta_i)$
    $sum\ over\ all\ path: v = \sum_{i=1}^{C} \alpha_i T_i(x)$
  $for\ all\ path\ i\ in\ the\ block: \beta_i \leftarrow \beta_i + \langle v, T_i(x) \rangle$
  return v

4 Implementation details

We adopt the flexible design of ResNets [3] and U-Net [4]. Our network consists mainly of 2 block: Up and Down Block (***Fig.*** *3*). We use the settings of 32 x 4d just as in [2] and starting layer has 32 filters. We use template 32x4x32 to express the network configuration.

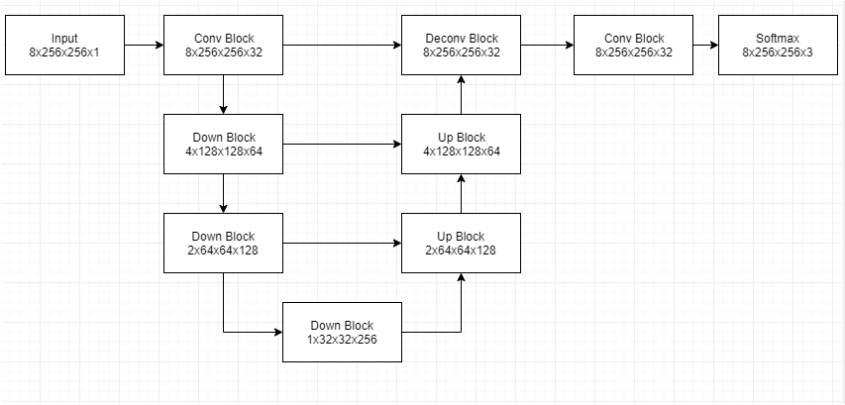

Fig. 3. Our ResCap model for kits19 challenge

We start the model with a standard convolution block with convolution layer followed by normalization layer and non-linear activation layer (*Fig. 4*)

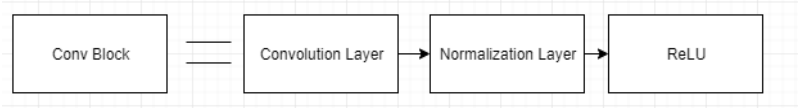

Fig. 4. Conv Block, standard convolution block with normalization and non-linear activation

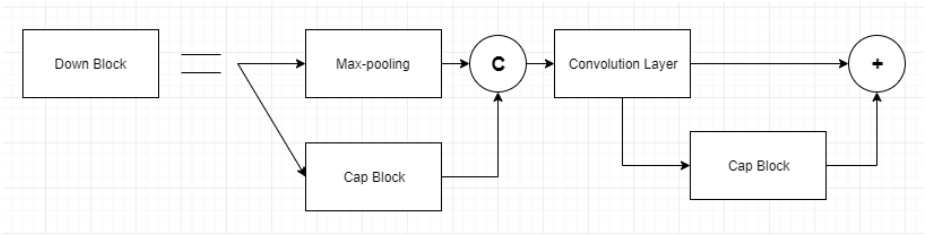

Fig. 5. Down Block, max-pooling layer concatenated with capsule block to pick out most prominent feature, followed by a refinement convolution layer with additional hierarchical information with capsule block

The Down block (***Fig.*** *5*) follows 3 simple rules: (i) any convolution/pooling layer must be accompanied by a capsule block (Figure 2), (ii) for pooling layer, capsule should be combined with it in a concatenative manner, and (iii) for convolution layer, capsule block should be combined with it in an additive manner.

For the Up block (Figure 6), we add one more rule: any transpose convolution layer must be combined with encoder's same resolution layer in a concatenative manner

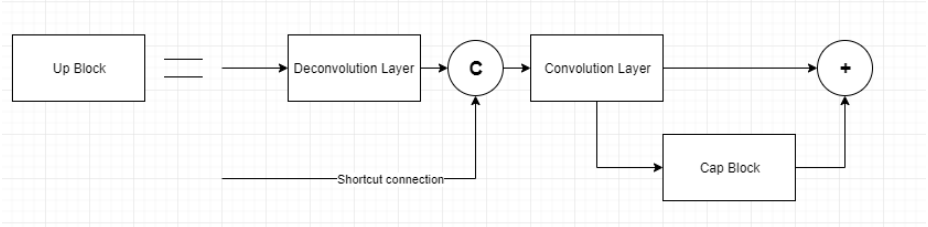

Fig. 6. Up Block, concatenate transpose convolution with shortcut connection from same resolution

Summation vs concatenation skip connection: for path that contains pooling operation or combination of transpose convolution and same resolution layer, we found it is empirically better to use concatenation. Concatenation is better than addition in our model, however, in order to stop the exponentially growth of number of parameters and GPU memory consumption, we settle for addition at same resolution layer and concatenation for changing resolution.

Normalization Layer: since the data consists of high resolution image (512 x 512), we use Group Normalization [4] instead of Batch Normalization [5] as we can only fit a small batch size on GPU. We found experimentally that adding a normalization layer after routing layer improved the model's performance. For big computation resource, Batch Normalization outperforms Group Normalization.

Routing vs pooling: In [1], the authors wanted to replace pooling layer with routing layer as they argued that pooling threw away local information like position and retained only the most active feature. We, however, think that the 2 need not be exclusive, as such, we include both max-pooling layer and routing layer in our network (***Fig.*** *6*). The intuition behind this is that while max-pooling does throw away much information, by using capsule as residual we can still retain the necessary information for higher layer to use. We found that for pooling and routing, concatenation works better than addition, we hypothesize this as capsule instantiation feature being overwhelmed by strongest feature picked out by max-pooling.

Softmax convolution vs capsule final layer: In segmentation task, there're a lot of empty background, and instantiation parameters are not well-defined for empty background, so using pure capsule output with additional instantiation feature doesn't make a lot of sense. By using softmax output, we side-step the issue of not well-defined instantiation feature and only focus on whether the foreground is there or not. Furthermore, each capsule output is independent of each other, but for segmentation task that has no overlapping object, we lose that information using capsule final layer.

5 Experiment

The goal of the dataset is to segment kidney and kidney tumor in CT scans of 300 unique kidney cancer patients. We train on 210 cases and test on the remaining 90 cases.

For this dataset, each case is a 3D CT scan of a patient, so by using normal 2D convolution we will lose 3D structure correlation. As such, we opted for 3D convolution on this dataset.
For each case in kits19 dataset, the scan resolutions aren't the same, so we standardize them all to 512x512 resolution by padding and cropping. After that, in order to fit the data on GPU without losing too much image information, we down sample the data to 256x256 using Bicubic Interpolation and extract a patch size of 8 along the z dimension. However, using only 8 frames along the z axis also resulted in much information loss and most of the tumor and kidney appear in the middle or near the end for each case, as such we add a normalized z coordinate as a channel for each of the slice.
Since the number of tumor and kidney slices is smaller than the ones without, we upsample them so that each batch has equal number of empty, tumor and kidney volume. Furthermore, since kidney is symmetrical with respect to the y axis, we augment the data with random horizontal flipping. After that, each volume is normalized to have zero mean and unit variance.
For loss function, we use Focal Loss [7] as it will automatically reduce learning signal of easy sample, so it'll act as a dynamic weighting scheme that helps improve our upsampling strategy.

$$\mathcal{L} = \frac{1}{n}\sum_{i=1}^{n} -(1-p_i)^{\gamma} Log(p_i) \tag{6}$$

$$\frac{\partial \mathcal{L}}{\partial p_i} = \frac{1}{n}\frac{(1-p_i)^{\gamma-1}}{p_i}\left(p_i - 1 + \gamma p_i Log(p_i)\right) \tag{7}$$

As we can see from Equ (7), the gradient magnitude reaches 0 as p_i approach 1 hence effectively reduce learning signal of easy sample. We found out experimentally that $\gamma = 1$ gives the best result.
Another dynamic weighted loss we tried was dice loss

$$\mathcal{L} = \frac{\sum_i p_i t_i}{\sum_i p_i^2 + t_i^2} \tag{8}$$

$$\frac{\partial \mathcal{L}}{\partial p_i} = \frac{t_i \sum_j (p_j^2 + t_j^2) - 2p_i \sum_j p_j t_j}{\sum_j \left(p_j^2 + t_j^2\right)^2} \tag{9}$$

As p_i goes to 1, it depends on how much other pixel reaches their target, so if other hard sample pixels still haven't reached their targets, easy samples pixel may still have large learning signal which results in not as effective weighting scheme as Equ (7).
For the final model, we train an ensemble of a network without z coordinate, a network with z coordinate with same number of parameter, and a network with z coordinate with double filter at each step. The results are showed in ***Table 1***.

Table 1 . Volumetric dice score for patch and case, 32x4x32 stands for 32 capsule path, 4 starting embedding, and start filter is 32, 32x4x64 stands for 32 capsule path, 4 starting embedding, and start filter is 64

Architecture	Patch dice score
Rescaps (32x4x32)	82.09%
Rescaps (32x4x32) + z coordinate	82.98%
Rescaps (32x4x64) + z coordinate	83.49%
Ensemble 3 models	83.82%

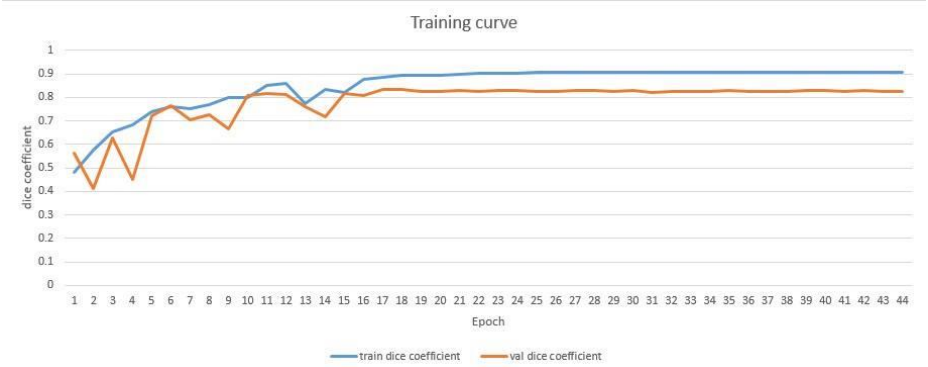

Fig. 7. Train and validation dice score curve

References

1. Sabour, S., Frosst, N., Hinton, G.E.: Dynamic Routing Between Capsules. (2017).
2. Xie, S., Girshick, R., Dollár, P., Tu, Z., He, K.: ResNeXt 2016 ImageNet 2nd: Aggregated residual transformations for deep neural networks. Proc. - 30th IEEE Conf. Comput. Vis. Pattern Recognition, CVPR 2017. (2017).
3. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (2016).
4. Wu, Y., He, K.: Group Normalization. Int. J. Comput. Vis. (2019).
5. Ioffe, S., Szegedy, C.: Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. (2015).