ReSet: Learning Recurrent Dynamic Routing in ResNet-like Neural Networks

Iurii Kemaev   Y.KEMAEV@GMAIL.COM
Skolkovo Institute of Science and Technology, Moscow, Russia
National Research University Higher School of Economics, Moscow, Russia

Daniil Polykovskiy   DANIIL.POLYKOVSKIY@GMAIL.COM
Moscow State University, Moscow, Russia
National Research University Higher School of Economics, Moscow, Russia

Dmitry Vetrov   VETROVD@YANDEX.RU
National Research University Higher School of Economics, Moscow, Russia

Editors: Jun Zhu and Ichiro Takeuchi

Abstract

Neural Network is a powerful Machine Learning tool that shows outstanding performance in Computer Vision, Natural Language Processing, and Artificial Intelligence. In particular, recently proposed ResNet architecture and its modifications produce state-of-the-art results in image classification problems. ResNet and most of the previously proposed architectures have a fixed structure and apply the same transformation to all input images.

In this work, we develop a ResNet-based model that dynamically selects Computational Units (CU) for each input object from a learned set of transformations. Dynamic selection allows the network to learn a sequence of useful transformations and apply only required units to predict the image label. We compare our model to ResNet-38 architecture and achieve better results than the original ResNet on CIFAR-10.1 test set. While examining the produced paths, we discovered that the network learned different routes for images from different classes and similar routes for similar images.

1. Introduction

Recent advances in Neural Networks made them prevalent in many problems of computer vision like image denoising, segmentation and captioning. Recently, they achieved and even surpassed a human-level performance in a wide range of tasks, including image classification and object detection He et al. (2016a); Russakovsky et al. (2015). Residual learning He et al. (2016a) is a building block for most of state of the art models. Deep Residual Networks showed outstanding results on the ILSVRC2015 challenge and quickly became one of the most promising architectures for many efficient models in computer vision.

Residual learning is based on the idea of shortcut connections between layers. If a standard layer learns some parametric mapping $y = f(x)$ of an input x, residual learning suggests to learn the mapping in a form of $y = x + f(x)$. The function f usually takes a form of a few convolutional layers with nonlinearities in between. The form of mapping $y = x + f(x)$ is motivated by the fact that very deep neural networks without shortcut connections

often exhibit worse performance than their shallower counterparts. Counterintuitively, as shown in He et al. (2016a), this result is due to underfitting, not overfitting. However, intuitively, deeper networks should not generalize worse, since last layers of the network can learn an identity mapping $y = x$, effectively learning a shallower model. Since a network cannot learn an identity mapping $f(x) = x$, adding a shortcut connection allows the latter layers to learn an identity mapping $y = x$ by setting $f(x) = 0$, which can usually be achieved by zeroing out all the weights. The idea of residual networks paved the way for deep models consisting of hundreds Huang et al. (2017b) and even thousands He et al. (2016b) of layers.

One of the most notable properties of ResNets is their robustness to deletion and reordering Veit et al. (2016); Srivastava et al. (2015) of layers. Randomly deleting or reordering of multiple layers of the trained ResNet does not significantly decrease the performance of the model. However, quality of non-residual models, like the VGG family Simonyan and Zisserman (2014), drops to the random prediction after these operations. This leads to the hypothesis that models with ResNet-based architectures exhibit ensemble-like behavior in the sense that their performance smoothly correlates with the number of layers. Veit et al. (2016) considered this effect as the result of an exponentially large number of paths of different lengths formed by skip connections. Although these paths are trained simultaneously, they are weakly dependent, and deleting some layers of the network affects only a small number of paths. From this point of view, authors proposed to treat ResNets as an exponential ensemble of shallow networks.

While studying this notable property, we assume that for each input object, only a small set of paths bring a significant contribution to the network's output. This set is determined by object's specificity, which sometimes can be interpretable. For example, a presence of certain objects on the picture may trigger the ResNet to transmit the information trough skip connections and not parametric transforms. This idea motivates and leads to the hypothesis that the number of required computations for inference in ResNets can be significantly reduced while saving or even increasing the efficiency of the network.

To detect this specificity, we propose a *Recurrent Set (ReSet)* module, which dynamically constructs a route for each input object through layers of a trained ResNet. This route can go through a single module multiple times or skip some modules. This routing is carried out by a ReSet's part called the *controller*.

Presumably, this scheme allows the network to learn a set of useful transformations and apply them in a case-specific order, that can lead to the better quality of classification and a lower number of required operations with a slightly greater number of parameters.

While we examined only a part of these insights, we hope that this experience will be useful for the scientific community and will help researchers to further deepen their understanding of ResNet-based models properties.

As our main contribution, we proposed the ReSet block that is indeed more flexible than SkipNet (Wang, 2017). However, SkipNet and ReSet have different motivations and outcomes—SkipNet's controller implicitly groups objects by complexity, while our model groups objects by their semantic similarity. For example, one can use routing scores as a short set of discrete image features (or embeddings). Also, SkipNet can only skip layers and cannot add new transformations, while ReSet can learn new features by combining learned layers and increase the quality. In our work, we described different approaches to ReSet's construction and provided many experiments with different settings. We believe that this

experience will be useful for future research on ResNets and neural networks with dynamic routing.

In this paper, we denote our contributions as follows:

- We propose a ReSet block that dynamically selects and applies transformations to the input data from a set of learnable layers (computational units).

- We design neural network ReSet-38 incorporating ReSet module into ResNet-38, and obtain better results on benchmarks while saving the same number of net's parameters

- We observed that proposed ReSet module groups objects in certain paths patterns by their semantic similarity (not only by complexity, as in described above models), which opens a wide range of possible applications of obtained path vectors (e.g. using them as image embeddings)

2. Related Work

Various recent studies on ResNets revealed many intriguing properties of neural networks equipped with skip connections.

Initially proposed ResNet is organized as three groups of layers called stages. Each stage is a sequence of a few blocks of layers with a skip connection. Between the stages, a network downsamples the image to reduce its spatial dimensions.

In addition to Veit et al. (2016) and Srivastava et al. (2015), authors of Greff et al. (2016) proposed another explanation of ResNets' robustness to deletion and reordering. Authors provided experiments showing that ResNet does not learn completely new representations within one stage, but instead its blocks refine the features extracted by the previous layer. Only after downsampling that is performed between successive stages, a new level of representation can be obtained. This process is called *unrolled iterative estimation* or an *iterative feature refinement* Jastrzebski et al. (2017). Later, Huang et al. (2017a) summed up the similarities between the boosting and an iterative feature refinement.

Considered properties of residual networks were implicitly exploited by many researchers in their works, resulting in great number of modifications of ResNet, which allows to significantly reduce the number of parameters and the number of operations during inference, without a decrease of performance.

Initially, Graves (2016) introduced Adaptive Computation Time (ACT) mechanism for recurrent neural networks. Since some objects are easier to classify than others, this mechanism dynamically selects the number of iterations for a recurrent neural network, while promoting the smaller number of iterations. Next, Figurnov et al. (2017) developed this idea further and incorporated ACT in ResNet-based architecture, where it decides whether to evaluate or to skip certain layers. They also introduced the Spacial Adaptive Computation Time (SACT) mechanism, which applies ACT to each spatial position of the residual block. Authors reported up to 45% reduction of the number of computations while preserving almost the same efficiency as original ResNets.

ShaResNet Boulch (2017) shares the weights of convolutional layers between residual blocks within one stage. This model is trained exactly in the same fashion as the original ResNet and reduces the number of parameters up to 40% without a significant loss in the

quality. Also, Jastrzebski et al. (2017) successfully reduced the number of weights three folds by sharing all residual blocks after the fifth within each stage. They investigated that sharing batch normalization statistics leads to low model efficiency, and resolved this issue by keeping a unique set of batch normalization statistics and parameters for each iteration.

SkipNet Wang et al. (2018) dynamically routes objects through a ResNet, skipping some layers. Authors developed a Hybrid Reinforcement Learning technique to reward the network for skipping blocks that have a small impact on the output. This way, they reformulated the routing problem in the context of sequential decision making, reducing the total number of computations on average by 40%. This work is similar to ours in the idea of dynamically selecting computational units. However, SkipNet can neither change the order of blocks nor select a single block more than once.

Finally, Leroux et al. (2018) used ideas of iterative estimation, adaptive computational time and weights sharing and proposed an Iterative and Adaptive Mobile Neural Network. The model is a compact network that consists of recursive blocks with ACT, and reduces both the size and a computational cost compared with ResNets. Authors maximally avoided redundancy using only one block, which recursively computes transformations until ACT module decides to proceed to the next stage.

3. Learning the Routing Policy

Following the work of Wang et al. (2018), we formulate the routing as a policy learning problem. We propose a model that at each iteration selects a Computational Unit (CU) F_{i_k} from a set of units $\mathcal{F} = \{F_1, \ldots, F_n\}$ and applies it to the data by computing

$$x_{k+1} = x_k + F_{i_k}(x_k). \tag{1}$$

The routing policy selects a Computational Unit by calculating a distribution vector $y_k \in \mathbb{R}^n$ that assigns a probability value for the selection of each Computational Unit. Each dimension corresponds to one of n available CU, where the higher value indicates the higher contribution of the corresponding unit. Then, we sample an index of a Computational Unit from the distribution y_k:

$$x_{k+1}^{hard} = x_k^{hard} + F_{i_k}(x_k^{hard}), \quad i_k \sim y_k \tag{2}$$

Since the selection of a computational unit is a non-differentiable function, we can consider a "soft" selection mechanism, that applies the policy by computing an expected output value with respect to the distribution y_k:

$$x_{k+1}^{soft} = x_k^{soft} + \sum_{j=1}^{n} y_{k_j} F_j(x_k^{soft}). \tag{3}$$

Notice, that when the weight y_{k_j} for a Computational Unit F_j is zero, evaluation of $F_j(x_i)$ can be skipped, saving computational time. With a slight abuse of notation, we further denote x_k^{soft} and x_k^{hard} as x_k, and the type of selection mechanism that is used will be clear from the context.

We compute the weights y_k using the *Controller* network C. The Controller takes a current representation of an object x_k and a current state of the controller h_k:

$$[\pi_k, h_{k+1}] = C(x_k, h_k), \tag{4}$$

where π_k are logits that can be transformed into a probability distribution by applying a Softmax function:

$$y_{k_i} = \frac{\exp(\pi_{k_i})}{\sum_{j=1}^n \exp(\pi_{k_j})}. \tag{5}$$

In the general case, the soft scoring for a dense vector y requires an evaluation of all CUs, requiring a lot of computational resources.

The soft selection is differentiable, but is more computationally expensive. Hard selection is a non-differentiable operation and requires some tricks to pass the gradient through sampling using REINFORCE algorithm, Gumbel trick, or Hybrid Reinforcement Learning.

3.1. REINFORCE

REINFORCE is a general-purpose algorithm for estimating gradient with respect to the policy Williams (1992), and is based on Monte-Carlo estimation of path-dependent gradients:

$$\nabla \mathbb{E}_y \mathcal{L} = \mathbb{E}_y \nabla \mathcal{L} \log y \approx \frac{1}{n} \sum_{t=1}^n \nabla \mathcal{L} \log y^t, \quad y^t \sim p(y) \tag{6}$$

Its known issue is notoriously high variance of gradients, which makes its application difficult. Several approaches to reduce variance were made in Greensmith et al. (2004). However, as experiments revealed, they appeared to be not powerful enough in the context of our problem. This result can be explained by an exponentially large number of routes that can be used for an object, requiring plenty of simulations in each pooling node for precise enough gradient estimation.

3.2. Gumbel-Softmax Estimator

Proposed in Jang et al. (2016), this technique performs continuous relaxation of one-hot discrete sampled vector:

$$y_i = \frac{\exp\left((\log(\pi_i) + g_i)/\tau\right)}{\sum_{j=1}^n \exp\left((\log(\pi_j) + g_j)/\tau\right)}, \tag{7}$$

where $\pi_1 \ldots \pi_n$ are logits for each option, τ is a constant called a *temperature*, and $g_1 \ldots g_n$ are samples from a Gumbel$(0,1)$ distribution, that can be easily computed as:

$$g_i = -\log(-\log(u_i)), \quad u_i \sim \text{Uniform}[0,1]. \tag{8}$$

This relaxation requires a soft selection. However, when a temperature τ tends to zero, weights of all elements but one become almost one-hot, effectively selecting only a single unit.

3.3. Straight-Through Gumbel-Softmax Estimator (ST)

Continuous relaxations of one-hot vectors still have no sparsity (although all components except one are close to 0). To sample one-hot vector, Jang et al. (2016) proposed to discretize y by taking $z = \text{one-hot}(arg\,max(y))$, and use continuous approximation in the backward pass by taking $\nabla_\theta y \approx \nabla_\theta z$. Thus, gradients become biased. Nevertheless, the bias approaches zero as the temperature tends to zero. This technique was called a Straight-Through (ST) Gumbel Estimator.

3.4. Hybrid Reinforcement Learning

After training the controller with sparse policy π (i.e. only one computational unit with non-zero score on each iteration), we propose to apply Hybrid Reinforcement Learning, introduced in Wang et al. (2018), to reduce number of CU's evaluations. For this, we add a separate CU that always returns zero. Combining with residual connection, selection of this unit will produce an identity mapping. To promote usage of this unit, we add a regularizer:

$$\min_{\theta,\pi} R_{\text{HybRL}}(\theta, \pi) = \mathbb{E}_y \sum_{k=1}^{t} R_k^{skip} y_k^{skip}, \tag{9}$$

where t is the total number of controller's invocations, k is the current one, R_k^{skip} is a reward for skipping the computation of a CU, i.e. using a zero unit. R_k^{skip} can depend on k or be a constant. y_k^{skip} is an indicator of whether computation was skipped or not. Maximizing this reward promotes the network to omit evaluation of CU, reducing the number of total operations.

4. ReSet

In this section, we describe the full architecture of dynamical routing that we call a *Recurrent Set* (ReSet) module.

The model consists of a *Controller* and a *pool (set) of Computational Units (CUs)*. For each object, the controller assigns scores to computational units and evaluates each block with a non-zero weight. A weighted sum of the results is added to the original representation, as described in Eq. 3. We repeat this operation a fixed number of times, and then pass the result to next layer.

ReSet module performs case-specific transformations of input objects, clustering their internal representations and passing them through different sets of computational units. Motivated with ResNet's robustness to deletion and shuffle of layers, the proposed recurrent structure has similarities with iterative refinement Greff et al. (2016). The proposed model may include multiple evaluations of the same computational unit, giving more flexibility than iterative refinement. The model with a soft selection is evaluated as follows:

$$[\pi_k, h_{k+1}] = C(x_k, h_k), \tag{10}$$

$$y_k = Softmax(\pi_k), \tag{11}$$

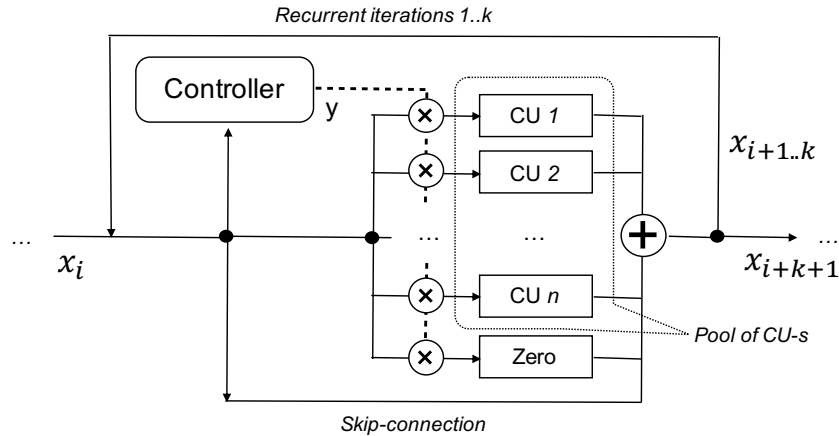$$x_{k+1} = x_k + \sum_{i=1}^{n} y_{k_i} F_i(x_k). \tag{12}$$

Figure 1: ReSet module. Input x_i is processed for k iterations. At each iteration, either a zero function, or one of n Computational Units (CU) is selected using a controller. The selected unit is evaluated with a value x_{i+j}, and the result is added to the current value of x_{i+j}, obtaining x_{i+j+1}. The final output of the model is x_{i+k+1}

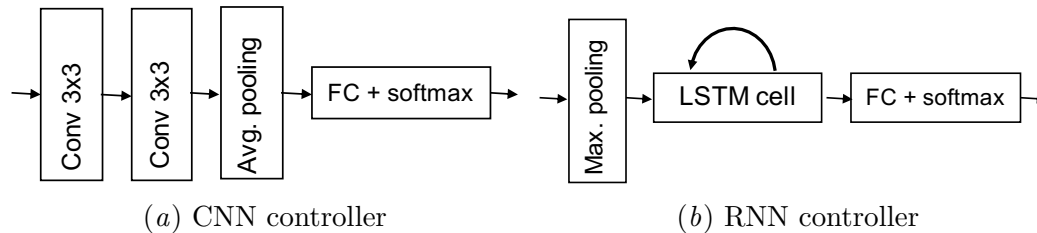(a) CNN controller $\qquad\qquad\qquad (b)$ RNN controller

Figure 2: Controller architectures. **a)** Convolutional controller. An input to this controller is a current feature representation x_i. **b)** Recurrent controller. Along with x_i, an intermediate LSTM layer stores the hidden state between representations.

We also add an entropy regularizer with of two terms: promoting different routes for different objects (high entropy across cases) and and promoting deterministic routes for each object on its own (low entropy for each case):

$$\min_{\theta} R_{ent}(\theta, y) = -\lambda_1 \sum_{i=1}^{t} \mathcal{H} \mathbb{E}_{\theta} y_i + \lambda_2 \sum_{i=1}^{t} \mathbb{E}_{\theta} \mathcal{H} y_i, \qquad (13)$$

where θ are model parameters, and $\mathcal{H}$ is entropy.

We tried two different architectures for a controller: a convolutional (CNN, Figure 2a) and a recurrent (RNN, Figure 2b) controller.

The *CNN controller* is a shallow convolutional network without biases with a global average pooling and a fully-connected layer with a Softmax in the end. As recommended in Jastrzebski et al. (2017), we store and recompute unique batch normalization statistics and parameters for each iteration. *RNN controller*, uses an LSTM cell Hochreiter and Schmidhuber (1997) to perform sequential decision making within one ReSet module. The diagram of a ReSet module is shown on Figure 1.
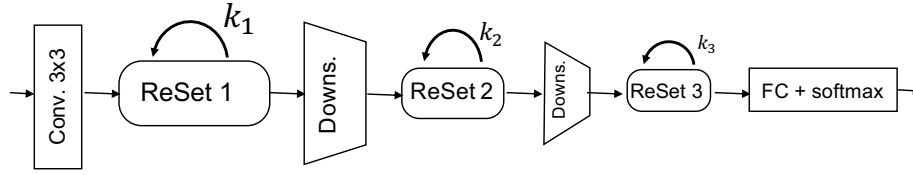
Figure 3: General scheme of ResNet with dynamic recurrent routing. Three stages of ReSet with intermediate downsampling layers and a final fully-connected layers with a Softmax.

Further, we designed a ReSet architecture (Figure 3), that is obtained by replacing stages in ResNet-38 with ReSet modules, allowing dynamic routing through a set of Computational Units presented by ResNet blocks. The model consists of 3 sequential parts:

1. A preliminary block containing a convolutional layer, followed by a batch normalization, ReLU, and a max pooling.

2. Three ReSet modules with a similar structure. Each module contains a pool of five Computational Units $\{F_1, \ldots, F_5\}$ and a zero-connection. Each CU is structured as original ResNet's block, i.e. small learnable network formed by two convolutional layers with batch normalizations and ReLU activations. We iterate each ReSet module five times with these CUs to produce the output. We also add an extra convolutional layer with a stride between the groups to downsample spatial dimensions.

3. Final global average pooling followed by a fully-connected classifier.

In general, ReSet's pool of Computational Units consists of n instances of a ResNet block. This decision allows to replace whole stage of original network with one ReSet module with certain number of iterations.

The algorithm for evaluating a ReSet block is shown in Algorithm 1. The whole architecture of the proposed models is shown in Figure 3.

Algorithm 1 ReSet transformation

Require:
   Controller C, Computational Units set F of size n, representation on i-th level x, number of iterations k
Ensure: x_{i+k+1} {representation on $i + k + 1$-th level}
   $x_i \leftarrow x$ {initialize result variable}
   $C.reset\_state()$ {reset controller's state}
   for $j \leftarrow 1, 2, \ldots, k$ **do**
      $y_j \leftarrow C.get\_scores(x_{i+j}, j)$ {get scores for CUs on current iteration}
      $x_{i+j+1} \leftarrow x_{i+j} + \sum_{r=1}^{n} y_{j_r} \cdot F_r(x_{i+j})$ {evaluate each CU from the pool and gather results weighted by scores}
   end for

Table 1: Results on CIFAR-10 (C10) and CIFAR-10.1 (C10.1) with shortened ReSet, $n_1 - n_2 - n_2$ shows the number of blocks on each of 3 stages.. Δ is the performance improvement with the corresponding baseline

Arc	Model	Acc, C10	Acc, C10.1	Gap	Acc, C10 Δ	Acc, C10.1 Δ
1-1-5	ResNet	88.7	**79.5**	9.2	-	-
	ReSet	**88.9**	**79.5**	9.4	0.2	0
1-5-1	ResNet	87	78.1	8.9	-	-
	ReSet	**88**	**78.7**	9.3	1	0.6
5-1-1	ResNet	86.5	78	8.5	-	-
	ReSet	**87.1**	**78.2**	8.9	0.6	0.2

5. Experiments

We used CIFAR-10 Krizhevsky and Hinton (2009) in provided experiments. It contains 50,000 images in the training set and 10,000 images in the test set, each of which has size 32×32 and is related to one of 10 classes. Also, to measure the effect of hyperparameters overfitting, we used an additional test set of images introduced in Recht et al. (2018); Torralba et al. (2008). We compare our results with the ResNet-38 architecture He et al. (2016a).

5.1. CNN controller vs RNN controller

To test the controller's ability to route objects, we conducted the following experiment. We took a pretrained ResNet-38 model, and put blocks from its stages into corresponding pools of computational units of three ReSet blocks. We fixed parameters of all computational units, and trained only a controller on 5,000 training batches of size 128 using Adam Kingma and Ba (2015) optimizer with a learning rate 0.001. We used He He et al. (2016a) initialization and a L2-regularizer on weights along with gradient clipping.

In a stated scenario, a good controller should find a deterministic policy corresponding to a sequential evaluation of a ResNet. We observed a close to uniform and ineffective policies of the CNN-controller, while a RNN-controller was able to find a sequential policy.

5.2. Learning ReSet-38 from scratch

To validate the idea of dynamic recurrent routing, we conducted experiments with shortened ResNet, that are shown in Table 1. From them it can be concluded that ReSet model found effective routing and better weights compared with original ResNet.

In the main part of experiments, we tried 3 different selection mechanisms (*scorers*) for controllers: Softmax, Gumbel Softmax and Gumbel Straight-Through estimator. We tried using Adam and SGD with Momentum optimizers with stepwise learning rate decrease (Adam: multiply by 0.95 every 1000 batches, SGD: as in original paper He et al. (2016a)). Also, we added an entropy regularizer to promote a selection of different routes by different objects. We varied the regularizer's weight, reducing it by the end of the training. Results are presented in the first section of Table 2.

Table 2: Results on CIFAR-10 and CIFAR-10.1. Default optimizer is SGD, SM – Softmax, GSM – Gumbel Softmax, GST – Gumbel Straight-Through estimator, ENTR – entropy regularizer promoting selection of different policies, HybRL – Hybrid Reinforcement Learning promoting selection of zero CUs. Contr. – training the only controller. Column with time indicates relative difference with baseline.

| N | Model | Accuracy | | Gap | Time Δ | Accuracy, Δ | |
		C10	C10.1			C10	C10.1
1	**ResNet38 (baseline)**	**92.8**	83.8	9	**-**	**-**	**-**
2	SM	91.9	83.5	8.4	1.92	-0.9	-0.3
3	SM, Adam	89.9	81.8	8.1	2	-2.9	-2
4	**SM+ENTR**	**92.1**	**85.1**	**7**	**1.92**	**-0.7**	**1.3**
5	SM+ENTR, Adam	89.3	79.5	9.8	2	-3.5	-4.3
6	GSM+ENTR	89	80.8	8.2	1.92	-3.8	-3
7	GST	60.5	54.1	6.4	1.08	-32.3	-29.7
8	SM pretr., Contr.	91.8	81.9	9.9	2	-1	-1.9
9	GSM pretr. SM, Contr.	91.8	83.5	8.3	2	-1	-0.3
10	GST pretr. SM, Contr.	66.5	54.1	12.4	1	-26.3	-29.7
11	GST pretr. GSM, Contr.	70.1	57.3	12.8	1	-22.7	-26.5
12	GST pretr. GSM	68.3	56	12.3	1	-24.5	-27.8
13	HybRL pretr GST	60.7	51.3	9.4	0.88	-32.1	-32.5
14	HybRL pretr GST	48.8	42.2	6.6	0.79	-44	-41.6
15	HybRL pretr GST	35.3	30.4	4.9	0.71	-57.5	-53.4
16	7 then finetune all	89.8	79.9	9.9	2.08	-3	-3.9
17	GST pretr., batch rep. x4	72.2	60.9	11.3	1.08	-20.6	-22.9
18	SM pretr., top-k 1	92.6	83.9	8.7	1.31	-0.2	0.1
19	SM pretr., top-k 2	92.7	84.1	8.6	1.92	-0.1	0.3
20	SM pretr., top-k 4	92.1	83.7	8.4	1.69	-0.7	-0.1

From these results we see that ReSet with a Softmax scorer and an entropy regularizer performs on-par with the original ResNet on the standard CIFAR-10 test set (used for validation here), outperforming on the CIFAR-10.1 test set. We decided to use only SGD with Momentum optimizer in further experiments because it outperformed Adam (with different hyperparameters) in almost all launches.

5.3. Pretraining and relaxation pipelines

Our attempts to train ReSet producing sparse scores from scratch have failed (*GST* in Table 2). We assume, that this can be explained by high variance of gradients, which is due to the large number of possible controller's choices (to be precise, $5^{5.3} \approx 3 \cdot 10^{10}$ choices). To optimize the whole architecture, we used the following relaxation pipeline:

1. take pretrained ResNet38/ReSet38 with Softmax
2. freeze all parameters except controller's (including batch normalization running statistics)

3. change controller's type to Gumbel Softmax (still not sparse, but closer to Gumbel-ST)

4. train controller on 5,000 batches

5. change controller's type to Straight-Through Gumbel Softmax

6. train controller until policy converged

7. add Hybrid Reinforcement Learning term to encourage reduction of operations

8. train the network until trade-off between efficiency and performance is reached

Results are presented in the second section of Table 2. We can see that the relaxation pipeline helps models to achieve better results compared to learning from scratch. The other important result is that while Hybrid Reinforcement Learning allows controlling computational cost, Gumbel ST estimator appeared to be too biased, leading to poor results.

5.4. Alternative techniques

We also tried several alternative techniques. The best-achieved results are presented in the third section of Table 2.

5.4.1. TWO-PHASE LEARNING

We separated the training procedure into 2 phases: controller's and CUs' learning, and then alternated training between them. Switching from one phase to another was based on several different rules: constant number of iterations, convergence of the current phase, progressive time for one of the phases.

This strategy did not give noticeable improvements, and straightforward procedures outperformed it. We think that the main reason of this is a permanently changing loss function surface for network's parameters, which led to the divergence of the learning process.

5.4.2. INCREMENTAL LEARNING

Take a pretrained network, freeze all its parameters, then unfreeze and train them incrementally, starting from last layers. In other words, unfreeze last layers, train the network on some batches, unfreeze last stage, train on another set of batches, unfreeze last but one stage, and so on.

The proposed modification has shown almost the same results as two-phase learning, and our conclusions are the same too.

5.4.3. TOP-K POOLS CHOICE (SOFTMAX STRAIGHT-THROUGH)

Compute scores with a Softmax-based controller and take the k largest of them, setting others to zero. In this case, computational units are getting "unequal" updates, which leads to unstable learning. Detailed analysis and comparison with other techniques can be found in Jang et al. (2016).

This technique showed almost the same results as the baseline. Noticeably, that efficiency is not a monotonous function of k. That can be explained by the local properties of the gradient descent.

Table 3: Results on CIFAR-10 and CIFAR-10.1 with shortened ReSet, $n_1 - n_2 - n_2$ shows the number of blocks on each of 3 stages. The scorer is Softmax.

N	Model	Accuracy		Gap	Time Δ	Accuracy, Δ	
		C10	C10.1			C10	C10.1
1	ResNet baseline, 2 CU & 2 iters	86.1	75	11.1	-	-	-
2	ReSet, 6 CU & 2 iter	87.2	76.9	10.3	**1**	1.1	1.9
3	**ReSet, 3 CU & 2 iters**	86.5	**78.4**	**8.1**	**1**	0.4	**3.4**
4	ReSet, 2 CU & 3 iters	86.4	77	9.4	1.08	0.3	2
5	**SM, 2 CU & 6 iters**	**87.3**	77	10.3	1.33	**1.2**	2

5.5. Additional experiments

We have also tried some additional experiment with computational pool's size and the number of iterations of ReSet layer. First, we used a shortened ReSet38 as a base model: we took two blocks (2 convolutions with batch normalization and ReLU) instead of the second and the third ReSet layers and modified the first ReSet layer as:

- 3 CUs in pool and 3 iterations (#pool = #iters)
- 2 CUs in pool and 6 iterations (#pool << #iters)
- 2 CUs in pool and 3 iterations (#pool < #iters)
- 6 CUs in pool and 2 iterations (#pool >> #iters)
- 3 CUs in pool and 2 iterations (#pool > #iters)

Results of the proposed modification and sequential model taken as a baseline are presented in Table 3. In this experiments, RNN-controller successfully exploits additional components and computational resources to outperform baseline model.

5.6. Examining learned routes

In this section, we visualize routing scores for a few images and analyze the obtained patterns. For each considered image, we found five images with the most similar routing pattern using the Manhattan distance. The results are shown in Figure 5. On Stage 1, distribution of scores collapses—presumably, the network extracts basic patterns, and on further stages, the distribution of paths (routes) has a much higher entropy (if there is no regularizer promoting a deterministic routing).

To demonstrate this, and that routing is, indeed, dynamic, we measured standard deviations of scores vectors' components produced by model Num. 8 in Table 2 for 10000 test CIFAR-10 images. The result is shown in Figure 4.

From these figures, it implies that ReSet's controller can distinguish pictures by some intrinsic semantic properties.

Analyzing the results, it becomes apparent that the network learned to pass images of different classes through different routes where similar by L1-metric paths are assigned to semantically similar images. Also, according to obtained patterns, routing between first stages is almost identical for all objects. However, the last layers used different routes for different classes, indicating that the network uses last iterations to refine its predictions.
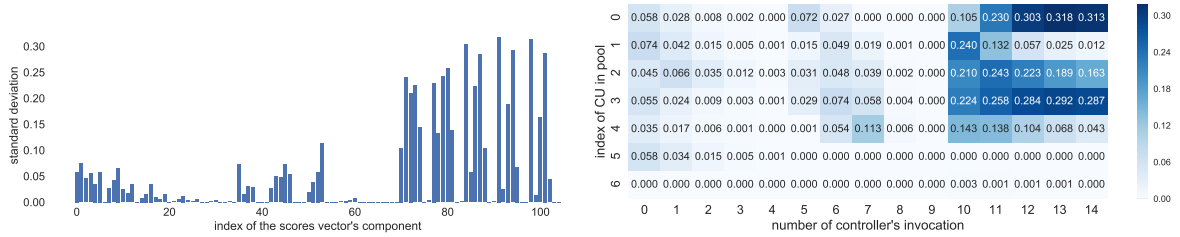
Figure 4: Standard deviations of scores vectors' components produced by model Num. 8 in Table 2 for 10000 test CIFAR-10 images. The standard deviation for certain components goes up to 0.3.
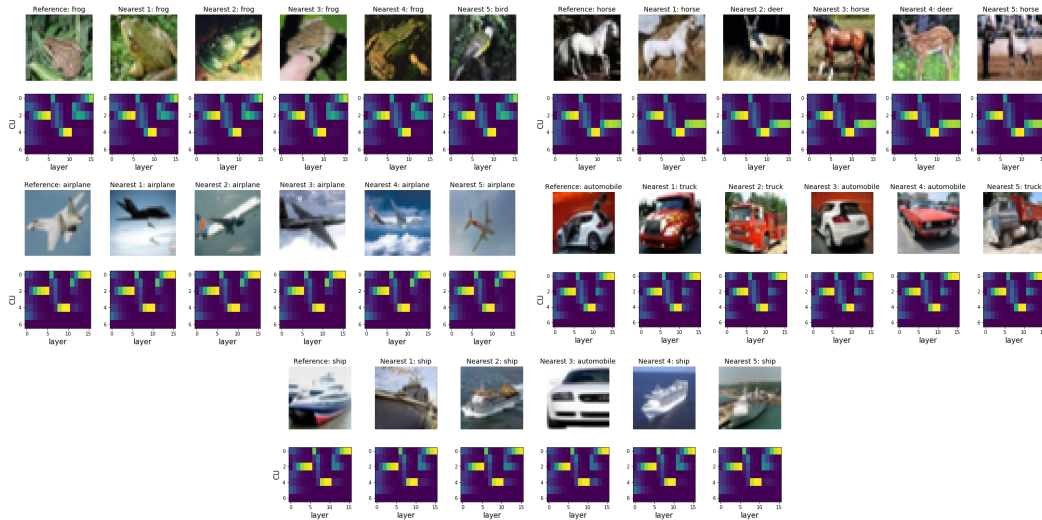

Figure 5: Routing through the network. For each experiment we show images that had the most similar routing distribution, along with the distribution itself.

5.7. Policy Evolution

In this section we visualize and analyze the evolution of policy for certain models. In Figure 6 we show average score of different Computational Units at different stages and iterations of the ReSet model.

From this figure, we can conclude that the learned policy can behave differently, depending on the set of the hyperparameters. For example, in Figure 6a, the learned policy was recurrent, since one block was selected at each iteration with almost certain probability. On stage 1, CU 2 learned to iteratively refine the result. In particular, the possibility to use a recurrent strategy was exploited in Leroux et al. (2018). Also, some stages develop a mixed behaviour, selecting different CUs for different objects. As suggested in the previous experiment, this usually happens at the last stage.

On Figure 6c and 6d we see that on stage 2, the policy is absolutely different from sequential (as in original ResNet-38), however, leading to the same result. This can be treated as new evidence of unrolled iterative estimation hypothesis Jastrzebski et al. (2017).

Noticeably, that entropy regularizer reduces the variance of scores (hence, the variance of gradients too) compared with model without it.

(*a*) Softmax-controller with entropy regularizer (Num. 3 in Table 2)

(*b*) Softmax-controller, a-priori scoring within stage

(*c*) Learning only controller, CU's are taken as blocks from pretrained ResNet (Num. 7 in Table 2)

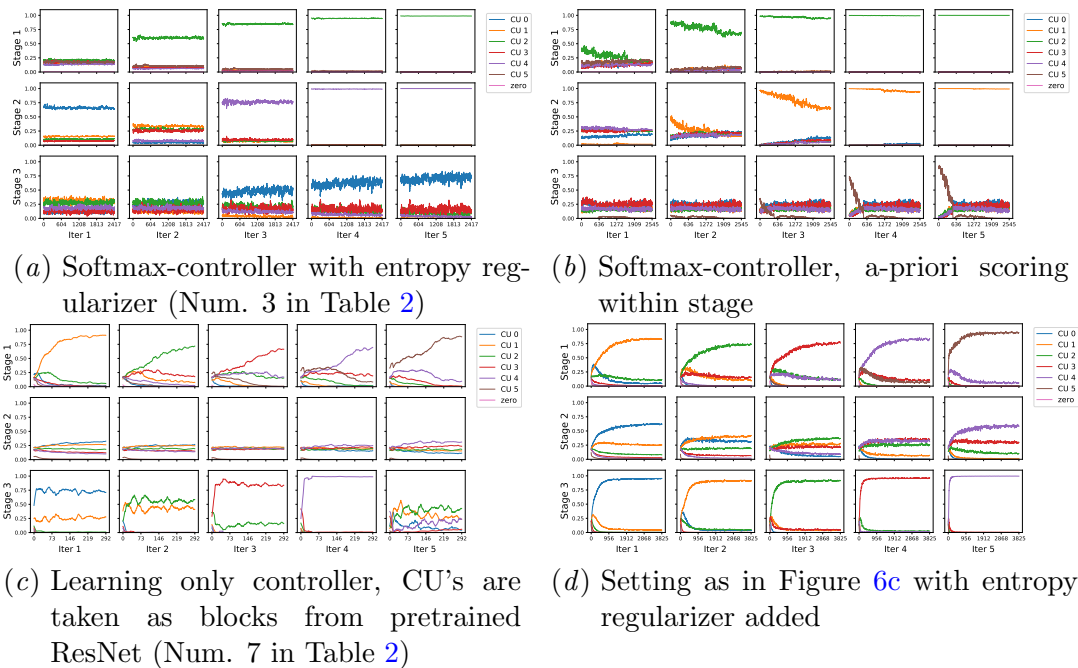(*d*) Setting as in Figure 6c with entropy regularizer added

Figure 6: Evolution of ReSet policies during training. Plots show the average score of the policy for each stage and each iteration inside the stage. An outer pair of axes describes a certain plot. In particular, an outer Y-axis shows the number of the stage number, an outer X-axis shows the number of iteration within that stage. An inner pair of axes describes the data on the plot: on inner Y-axis we placed the mean of controller's scores for certain CU, and inner X-axis shows the number of processed during training batches (in thousands, each 25-th measurement for compactness).

6. Conclusion

In this work, we introduced a ReSet layer that performs dynamic routing through the set of independent Computational Units (transformations). The proposed model achieved better classification results compared with the ResNet-38 model, having a comparable number of parameters. The model learned to separate paths of images from different classes and produced separate Computational Units for the final stage of the network to refine its predictions.

The obtained results open a wide range of possible applications of the proposed mechanism of dynamic recurrent routing with ReSet. For example, ReSet could be used in Natural Language Processing, where one would expect the ReSet to process different parts of the sentence with different Computational Units. An additional direction of research is the properties of image's controller scores vectors, which could be considered as corresponding embeddings for pictures.

Acknowledgments

References

Alexandre Boulch. Sharesnet: reducing residual network parameter number by sharing weights. *arXiv preprint arXiv:1702.08782*, 2017.

Michael Figurnov, Maxwell D Collins, Yukun Zhu, Li Zhang, Jonathan Huang, Dmitry P Vetrov, and Ruslan Salakhutdinov. Spatially adaptive computation time for residual networks. In *CVPR*, volume 2, page 7, 2017.

Alex Graves. Adaptive computation time for recurrent neural networks. *arXiv preprint arXiv:1603.08983*, 2016.

Evan Greensmith, Peter L Bartlett, and Jonathan Baxter. Variance reduction techniques for gradient estimates in reinforcement learning. *Journal of Machine Learning Research*, 5(Nov):1471–1530, 2004.

Klaus Greff, Rupesh K Srivastava, and Jürgen Schmidhuber. Highway and residual networks learn unrolled iterative estimation. *arXiv preprint arXiv:1612.07771*, 2016.

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, 2016a.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *European Conference on Computer Vision*, pages 630–645. Springer, 2016b.

Sepp Hochreiter and Jurgen Schmidhuber. Long short-term memory. *Neural computation*, 9 8:1735–80, 1997.

Furong Huang, Jordan Ash, John Langford, and Robert Schapire. Learning deep resnet blocks sequentially using boosting theory. *arXiv preprint arXiv:1706.04964*, 2017a.

Gao Huang, Zhuang Liu, Kilian Q Weinberger, and Laurens van der Maaten. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, volume 1, page 3, 2017b.

Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*, 2016.

Stanisław Jastrzebski, Devansh Arpit, Nicolas Ballas, Vikas Verma, Tong Che, and Yoshua Bengio. Residual connections encourage iterative inference. *arXiv preprint arXiv:1710.04773*, 2017.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *International Conference for Learning Representations*, 2015.

Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. 2009.

Sam Leroux, Pavlo Molchanov, Pieter Simoens, Bart Dhoedt, Thomas Breuel, and Jan Kautz. Iamnn: Iterative and adaptive mobile neural network for efficient image classification. *arXiv preprint arXiv:1804.10123*, 2018.

Benjamin Recht, Rebecca Roelofs, Ludwig Schmidt, and Vaishaal Shankar. Do cifar-10 classifiers generalize to cifar-10? 2018. https://arxiv.org/abs/1806.00451.

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3): 211–252, 2015.

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

Rupesh Kumar Srivastava, Klaus Greff, and Jürgen Schmidhuber. Highway networks. *arXiv preprint arXiv:1505.00387*, 2015.

Antonio Torralba, Rob Fergus, and William T. Freeman. 80 million tiny images: A large data set for nonparametric object and scene recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30(11):1958–1970, 2008.

Andreas Veit, Michael J Wilber, and Serge Belongie. Residual networks behave like ensembles of relatively shallow networks. In *Advances in Neural Information Processing Systems*, pages 550–558, 2016.

Xin Wang, Fisher Yu, Zi-Yi Dou, Trevor Darrell, and Joseph E. Gonzalez. Skipnet: Learning dynamic routing in convolutional networks. In *The European Conference on Computer Vision (ECCV)*, September 2018.

Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. In *Reinforcement Learning*, pages 5–32. Springer, 1992.