A Reproduction of Residual Attention Network for Image Classification

E4040.2021Fall.IEEE.report
Jerry Liu jl6007, Nathan Wu yw3731, Qinghang Hong qh2249
Columbia University

Abstract

In this project, we replicate the Residual Attention Network[1] on TensorFlow 2.0 and conduct experiments on CIFAR-10 and CIFAR-100[2]. Our initial goal is to achieve similar results on fitting the CIFAR-10 dataset in the paper, which gives a 94.11% validation accuracy. With our implementation of Attention-56 network, we have reached 88.10% accuracy with hyper-parameter optimization and data augmentation. Our results are fairly close to the paper, having only a 6.01% validation accuracy difference. Training on a relatively larger dataset, e.g., CIFAR-100 dataset serves as an additional goal that we want to achieve. The paper does not conduct experiments on CIFAR-100 with Attention-56, but we have achieved 83.4% training accuracy and 64.94% validation accuracy. One of the main technical challenges we have met is the problem of overfitting, and we have solved it by augmenting the dataset and compressing the model. This project significantly shows the effectiveness of Residual Attention Network and provides a detailed observation on the solutions to overfitting problems.

1. Introduction

Attention mechanism[3] has been proved effective and widely used in multiple fields of machine learning, including computer vision and natural language processing.

The attention mechanism can be considered to increase the model capacity and highlight the salient features of the image. The goal of this project is to replicate the image classifier Residual Attention Network, which incorporates attention with pre-activation Residual Unit[4]. We want to reproduce similar performance results on CIFAR-10 and CIFAR-100 datasets as stated in the original paper.

2. Summary of the Original Paper

2.1 Methodology of the Original Paper

The Residual Attention Network mainly consists of several Attention Modules that are connected to each other. In Figure 2. Each Attention Module is using a pre-activated Residual Unit[4], which can be also replaced by other convolutional network architecture, as the basic residual unit. Followed by multiple residual units with output *x*, the Attention Module is divided into two branches: trunk branch and mask branch. The trunk branch is constructed by simply stacking up several residual units, acting as a feature processing block and outputting *T(x)*. There are 36m+20 weighted layers in the trunk branch, where m is the number of Attention Modules in one stage.

The mask branch uses bottom-up top-down architecture to produce sigmoid activated attention/mask score *M(x)* that is used to re-weights *T(x)*, aiming to focus on the important features by giving them higher weights. In the mask branch, max pooling is performed multiple times to downsample the feature map and increase the receptive field. Then after some residual units, the same number of upsampling with bilinear interpolation is used to keep the same output size as the input feature. Similar to residual learning in ResNet[5], the authors use element-wise multiplication

$$H(x) = (1 + M(x)) * T(x)$$

to combine the outputs of two branches, where the *M(x)* and *T(x)* have the same shape, to preserve the majority of features *T(x)* and prevent gradient vanishing. After producing a masked enhanced feature *H(x)*, the authors then feed *H(x)* into several residual units.

We set the number of Attention Modules in each stage m = {1, 2, 3, 4}. For the Attention Module, the authors name these models Attention-56, Attention-92, Attention-128 and Attention-164 respectively.

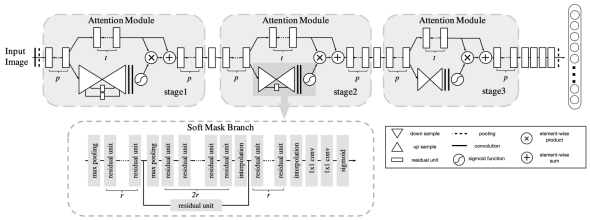

Figure 2: Example architecture of the proposed network for ImageNet. We use three hyper-parameters for the design of Attention Module: p, t and r. The hyper-parameter p denotes the number of pre-processing Residual Units before splitting into trunk branch and mask branch. t denotes the number of Residual Units in trunk branch. r denotes the number of Residual Units between adjacent pooling layer in the mask branch. In our experiments, we use the following hyper-parameters setting: $\{p = 1, t = 2, r = 1\}$. The number of channels in the soft mask Residual Unit and corresponding trunk branches is the same.

2.2 Key Results of the Original Paper

The authors conduct experiments on datasets including CIFAR-10, CIFAR-100, and ImageNet and compare results with some alternative designs of the model. They

also compare the performance and model size of Residual Attention Network with some state-of-the-art architectures to prove effectiveness and efficiency.

The residual attention network, Attention-452 achieves 3.90% and 20.45% error rate on CIFAR-10 and CIFAR-100 respectively, outperforming other state-of-the-art models including ResNet-164[4], WRN-16-8[7] with fewer parameters in Table 6.

Network	params$\times 10^6$	CIFAR-10	CIFAR-100
ResNet-164 [11]	1.7	5.46	24.33
ResNet-1001 [11]	10.3	4.64	22.71
WRN-16-8 [39]	11.0	4.81	22.07
WRN-28-10 [39]	36.5	4.17	20.50
Attention-92	1.9	4.99	21.71
Attention-236	5.1	4.14	21.16
Attention-452†	8.6	**3.90**	**20.45**

Table 6: Comparisons with state-of-the-art methods on CIFAR-10/100. †: the Attention-452 consists of Attention Module with hyper-parameters setting: $\{p = 2, t = 4, r = 3\}$ and 6 Attention Modules per stage.

Attention-56, Attention-92 are also trained and evaluated on ImageNet and reach similar or better performance with much fewer model parameters in Table 7.

Network	params$\times 10^6$	FLOPs$\times 10^9$	Test Size	Top-1 err. (%)	Top-5 err. (%)
ResNet-152 [10]	60.2	11.3	224×224	22.16	6.16
Attention-56	31.9	6.3	224×224	**21.76**	**5.9**
ResNeXt-101 [36]	44.5	7.8	224×224	21.2	5.6
AttentionNeXt-56	31.9	6.3	224×224	**21.2**	**5.6**
Inception-ResNet-v1 [32]	-	-	299×299	21.3	5.5
AttentionInception-56	31.9	6.3	299×299	**20.36**	**5.29**
ResNet-200 [11]	64.7	15.0	320×320	20.1	4.8
Inception-ResNet-v2	-	-	299×299	19.9	4.9
Attention-92	51.3	10.4	320×320	**19.5**	**4.8**

Table 7: Single crop validation error on ImageNet.

For ablation studies, the authors compare the performance of two different kinds of residual learning for integrating the trunk and mask branch. They propose naive attention learning(NAL), where the output is computed:

$$H(x) = M(x) * T(x)$$

and attention residual learning(ARL), where the output is computed as:

$$H(x) = (1 + M(x)) * T(x)$$

The experiments prove that ARL shows significant performance boost compared to NAL in Table 3.

Network	ARL (Top-1 err. %)	NAL (Top-1 err.%)
Attention-56	**5.52**	5.89
Attention-92	**4.99**	5.35
Attention-128	**4.44**	5.57
Attention-164	**4.31**	7.18

Table 3: Classification error (%) on CIAFR-10.

3. Methodology (of the Students' Project)

We choose the pre-activated residual unit as our basic residual unit block, as illustrated in Fig 3. It consists of batch normalization, convolution, ReLU and residual skip connections.

In our model, we have designed 3 stages with one attention module in each stage. In the i-th stage (0-indexed), we perform max pooling 2-i times, with 2 residual units followed by each max pooling. Then we have 2 residual units followed by each upsampling with linear interpolation. We also add residual skip connections between k-th max pooled feature map and last k-th upsampled feature map for $k \neq 0$. The model architecture is identical to Attention-56 in the original paper.

After three stages of attention modules, we add three residual units followed by an average pooling layer, which is also consistent with the original architecture.

Lastly, we flatten the layer and add a feedforward layer of x nodes with softmax activation, where x is the number of classes depending on the dataset.

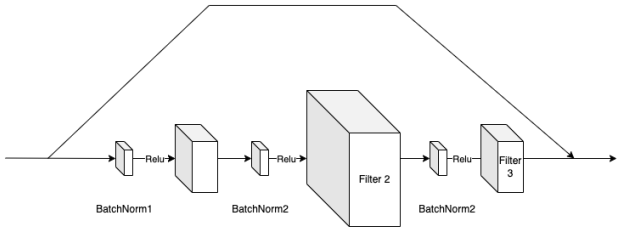

Figure 3

3.1. Objectives and Technical Challenges

The objective of this project is to replicate the residual attention network architecture *Attention-56*, train it on CIFAR-10, and try to produce similar performance as the results stated in the original paper. Besides, training on the CIFAR-100 dataset serves as an additional goal in our schedule.

3.2.Problem Formulation and Design Description

Image classification is a problem mapping an image to its class. Mathematically, an image classifier f

$$f: X \rightarrow Y$$

where X is an image data, a matrix of shape (height, width, channel) and $Y \in Label$. An image is often RGB with three channels, or grey-scaled with one channel. Both types of images have elements $x \in [0, 255]$. We can also have a mapping from label to an index, which can help to digitalize the label and formulate mathematical models.

4. Implementation

In this section, we will describe the datasets we use in 4.1, deep learning network training in 4.2. We will describe code architecture of the project and provide an overall pipeline of the training process.

4.1 Data

We use CIFAR-10 and CIFAR-100[2] as our datasets for experiments. CIFAR-10 dataset contains 60,000 RGB images of size 32*32, divided into 50,000 training images and 10,000 test images. It has 10 different classes and 6,000 images of each class. CIFAR-100 has the same number of images of the same size, but it has 100 classes and 600 images of each class. Each image of the datasets contains items of exactly one class. For example, there are no images with both birds and cats in it.

For data pre-processing, we use feature wise center and standard normalization in both training and testing data. To make the model more robust, we use data augmentation, including random shifts in both height and width and randomly horizontally flip some images during training.

4.2 Deep Learning Network

We use Adam optimizer with batch size of 64 and an initial learning rate of 0.0001. We then adopt learning rate reduction techniques, multiplying a factor of 0.2 on learning rate if validation accuracy does not improve for 7 consecutive epochs. We train the model for 200 epochs, equivalent to 160k iterations in the original paper. We also use early stopping if the validation accuracy does not improve for 15 consecutive epochs. We use *categorical cross entropy loss* as the loss function and *accuracy* as performance metric. The top-1 error rate can also be computed from accuracy as (1 - top-1_error_rate).

We adopt the Attention-56 structure in the original paper, with 3 stages and one attention module in each stage. We use filters=[x_i,x_i, $4x_i$] for residual units in stage i, with x=[32, 64, 128]

4.3 Software Design

Our code repository is in [7]. The top level flow chart is shown in Fig 7. The pipeline follows the standard image classification process. We use tensorflow API to load the image, transform and augment data with ImageGenerator and build our own residual attention model. Then we proceed to define necessary components, such as optimizer, loss and performance metric. After preparing all ingredients, we use TensorFlow API fit() to train the model. Upon finishing the training model, we save the model weights and plot the performance history.

We divide our code into 3 main sections, trainer, model.py, layer_funcs.py. layer_funcs.py contains the util layers, residual unit, attention module and attention learning function, which are needed to build our model. We then build the model in model.py using the essential components in layer_funcs.py. We have two jupyter notebooks of the same structure for trainers, with each one corresponding to training CIFAR-10 and CIFAR-100 respectively.

The detailed function calling graph of model building is illustrated in Figure 8, where A→ B means that function A needs to call function B inside the function A definition.

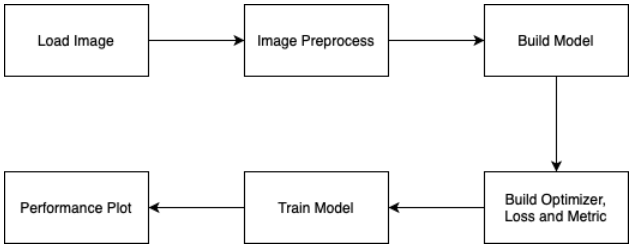

Figure 7

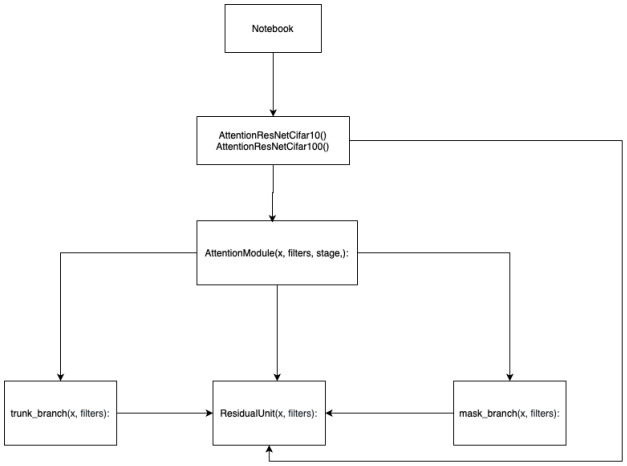

Figure 8

5. Results

5.1 Project Results

For CIFAR-10, we reach a 88.10% testing accuracy with hyperparameters optimization and data augmentation. The training is early stopped at epoch 80. It takes a total of 6.4 hours to train using a single GPU. The trend of training accuracy and validation accuracy is

shown in Fig 4. And the trend of training loss and validation loss is shown in Fig 5. The training and testing accuracy steadily increases during training, and reaches a plateau in the 50th epoch. The training and testing losses steadily decrease until the 50th epoch.

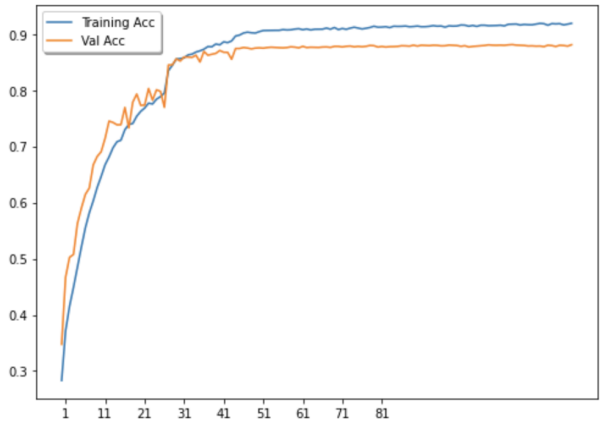

Figure 4

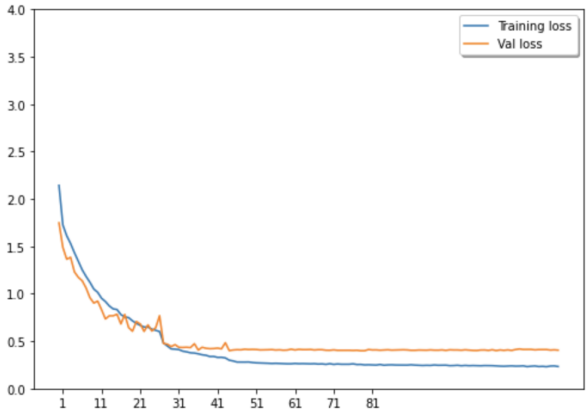

Figure 5

For CIFAR-100, we also use *Attention-56* architecture as mentioned above. The model reaches 83.4% training accuracy and 64.94% testing accuracy with hyperparameters optimization and data augmentation. The training is early stopped and it takes a total of 84 epochs. For training time consumption, it takes about 4 hours to achieve 65% validation accuracy and 6.5 hours before the early stop. The history of training and validation accuracy is shown in Fig 9. The trend of training and validation losses is shown in Fig 10. Note that the model reaches a plateau around the 45th epoch.

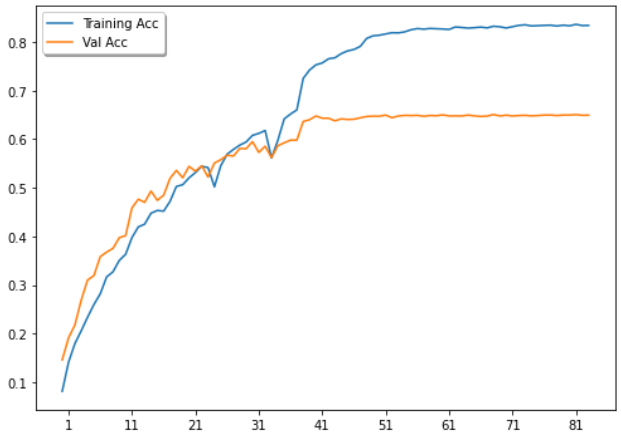

Figure 9

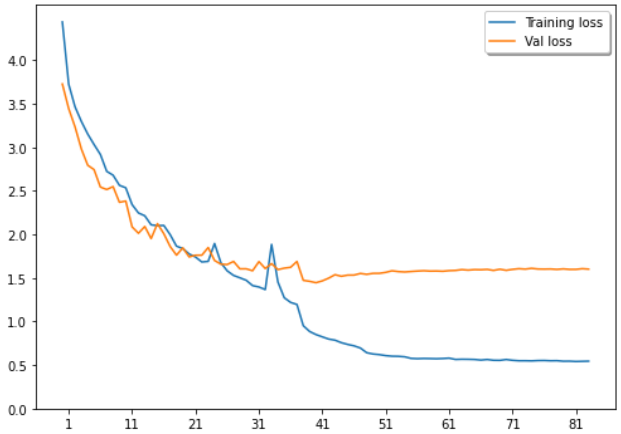

Figure 10

5.2 Comparison of the Results Between the Original Paper and Students' Project

Limited by available computing resources and time for training, we only trained our models on CIFAR-10 and CIFAR-100 data, and obtained 88.10% and 65.59% validate accuracy respectively (Table 5.2-1). In the paper, however, the model has only been trained on the CIFAR-10 dataset and obtained an accuracy of 94.11%. We got a fairly close result, having only a 6.01% validation accuracy difference, compared to the one in the paper. In terms of the training duration, we trained our model using a single GPU over 130 epochs, while the papers spent 160k (steps)/780(step_per_epoch) ≅ 200(epochs) on training (Figure 6).

	Our Result	Paper's Result
CIFAR-10	88.10%	94.11%
CIFAR-100	64.94%	N/A

Table 5.2-1

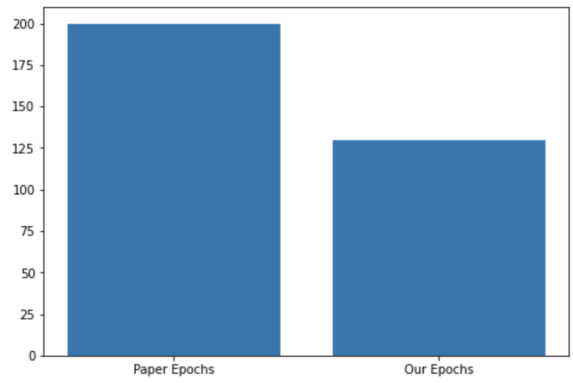

Figure 6

5.3 Discussion of Insights Gained

Initially, we used the hyper-parameters provided by the original paper, including learning_rate=0.1, momentum=0.9 and nesterov SGD optimizer. However, we ran into the problem that loss and output logits become NaN, which was due to the learning rate being too large. Then we changed the optimizer into Adam with the reducing learning rate starting from 0.0001 and solved the problem. We found data augmentation playing a huge role in performance boosting. Initially, we only reached 75.32% validation accuracy while almost 100% training accuracy for the original CIFAR-10 dataset. Later after implementing training data augmentation, we reached 88.10% validation accuracy, which solved the overfitting problem. We also found the hyperparameter tuning was crucial in training a model on different datasets. Different sets of hyperparameters could bring drastically different results, perhaps giving larger effects on the performance than some innovative design of the models. Additionally, each dataset could require different sets of optimal hyper-parameters.

6. Future Work

We hope to further our work in these following directions. 1) train on larger dataset, such as ImageNet. 2) conduct ablation studies on our current model by taking out some critical components such as the softmask branch to evaluate the effectiveness of each component.

7. Conclusion

In this project, we replicated the Residual Attention Network and trained it on CIFAR-10 and CIFAR-100 data. Specifically, we rebuilt the *Attention-56* network and achieved similar validation accuracy (with only 6.01% difference) on the CIFAR-10 data as in the paper. The goals of this project, reproducing Residual Attention Network results with limited computational resources, were successfully achieved. Besides, we fitted the *Attention-56* model on the CIFAR-100 dataset and also obtained a decent accuracy. In addition to reproducing the model, we have made some interesting and significant observations along the way: we have tested the effectiveness of data augmentation in avoiding overfitting and have witnessed how network structures and hyperparameters could greatly affect the performance.

6. Acknowledgement

We would like to express our special gratitude to our ECBM 4040 instructor Dr. Kostic and all the TAs for their guidance and help. We have practically utilized the theoretical knowledge about hyperparameters tuning and solutions to reduce overfitting. We appreciate the TensorFlow framework that has provided us with neat APIs to build the model. We also appreciate users of GitHub and StackOverflow, who have shared their brilliant solutions to the common errors. They made our development as smooth as possible.

7. References

[1]F. Wang et al., “Residual Attention Network for Image Classification,” arXiv:1704.06904, Apr. 2017.

[2]A. Krizhevsky, “Learning Multiple Layers of Features from Tiny Images,”.

[3]V. Mnih, N. Heess, A. Graves, and koray kavukcuoglu, “Recurrent Models of Visual Attention,” in Advances in Neural Information Processing Systems, 2014, vol. 27.

[4]K. He, X. Zhang, S. Ren, and J. Sun, “Identity Mappings in Deep Residual Networks,” arXiv:1603.05027, Jul. 2016.

[5]K. He, X. Zhang, S. Ren, and J. Sun, “Deep Residual Learning for Image Recognition,” arXiv:1512.03385, Dec. 2015.

[6]S. Zagoruyko and N. Komodakis, “Wide Residual Networks,” arXiv:1605.07146, Jun. 2017.

[7]https://github.com/ecbme4040/e4040-2021fall-project-ieee-qh2249-yw3731-jl6007

8. Appendix

8.1 Individual Student Contributions in Fractions

	yw3731	jl6007	qh2249
Last Name	Wu	Liu	Hong
Fraction of (useful) total contribution	1/3	1/3	1/3
What I did 1	Write residual unit	Write attention module	Write model architecture
What I did 2	Train CIFAR-10	Train CIFAR-100	Write training process
What I did 3	Write report	Write report	Write report