Residual Attention Network for Image Classification

E4040.2020Fall.RAN.report
Hangyu Tian - HT2459, Jiaxiang Wang - JW3864
Columbia University

Abstract

In this work, we intend to study and understand the 'Residual Attention Network' proposed by Wang et al. [1] by reconstructing the model closely following the model description as well as validating most of the experiment results stated in the paper [1]. Our model training and experiments are conducted on CIFAR-10 and CIFAR-100 datasets, and due to the lack of mention of few network designs specific to these datasets, the extended experiment is done for us to match the stated model performance. The same datasets are used to conduct various experiments proposed by Wang et al. to validate the effectiveness of each RAN building components. Despite our persistent effort, our performance results on the network cannot match the result stated in the paper. (CIFAR-10 14.43% error vs. 3.9% error from paper, CIFAR-100 49% error vs. 20.45% error from paper) However, our experiments are able to confirm each RAN component introduced by Wang et al. contributes effective improvement to the performance of the model.

1. Introduction

Ever since the successful debut of ResNet [4] in 2015, the concept of residual learning framework has gained popularity among the computer vision world due to its simple implementation and ease and effectiveness of training in deep convolutional networks [2, 3, 4]. Meanwhile, the attention mechanism, while first proposed by Bahdanau et al. [5] in 2015 as an optimizing mechanism to the encoder-decoder neural machine translation model, at the time of the original paper's writing it has only stayed in the realm of sequence generation and not being grandly introduced into feedforward structure classification tasks such CNN image classification. [6] With the proposal of Residual Attention Network by Wang et al. [1] in their paper, the authors bring the two concepts together and achieve superior performance over the state-of-the-art image classification methods. We are fascinated by the impressive outcome resulting from the combination of two cutting edge deep learning concepts and decide to study its essence in a thorough manner.

We begin our study approach by first recreating the Residual Attention Network proposed in the paper. Our goal for this step is to match the performance result stated in the paper meanwhile following the proposed model design as closely as possible. After we conclude our first objective, we conduct extended analyses on building components such as Mask Branch and Residual Attention Learning, with aim to not only validate the results from the original paper but also to thoroughly understand the mechanism and effectiveness of each building component of the network.

Given limited computational resources, we are unable to replicate the performance result for the ImageNet datasets, therefore we decide to only conduct our training and experimentations using CIFAR-10 and CIFAR-100 datasets. When developing the model for such datasets, we face several challenges related to the model design. Although the paper offers a detailed description for each building component, which allows us to replicate the structure of the proposed model almost completely, there are few design settings missing for CIFAR designated networks. To overcome these challenges, we conduct extended testing on these settings to bring the error rate down to the appropriate level.

2. Summary of the Original Paper

2.1 Methodology

The original paper can be broadly separated into two sections based on the overall objective of the content. In the first section, which is viewed as the introduction of Residual Attention Network, the authors briefly talk about the motivation of the network with its appealing properties. They then introduce in detail the structure and the underlying effect of each building components used to create the network. In the second section, which is viewed as the performance evaluation section, the authors first conduct extended experiments using CIFAR-10 and CIFAR-100 datasets to test out the effectiveness of the building components. Then they test out robustness of the network by injecting random noise to the training data and comparing the performance with that of the state-of-the-art methods ResNet-164 [4]. After discussing the results, the network and all its varied versions are compared with state-of-the-art methods such as ResNet [2] and Wide ResNet [3]. A similar performance evaluation is also conducted using the ImageNet dataset.

2.2 Key Results

There are two broad levels of conclusions drawn by the original paper. The top-level conclusion states the superior performance of the Residual Attention Network over states-of-the-art image classification methods on benchmark datasets, *i.e.*, ResNet on CIFAR-10 (3.90% error), CIFAR-100 (20.67% error), and challenging

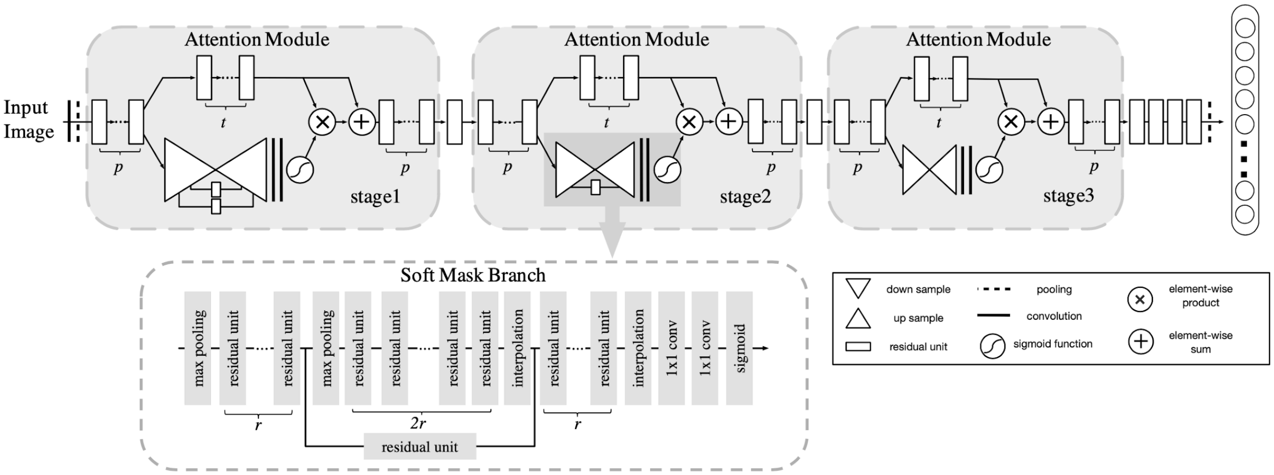


Figure 1: Architecture of the proposed network for ImageNet from the Original paper. hyper-parameters setting: {p = 1, t = 2, r = 1}

ImageNet dataset (0.6% top-1 accuracy improvement) with only 46% trunk depth and 69% forward FLOPs (comparing with ResNet-200). The lower level of conclusion confirms the following three statements:

1. Free form mixed attention will have better performance than constrained (including single) attention.
2. Encoding top-down attention mechanism into bottom-up top-down feedforward convolutional structure in each Attention Module allows the network to be extensible.
3. Residual attention learning allows training very deep Residual Attention Network.

These three statements, as validated by the extended experiments, act as the supporting factors to the competitive performance of the Residual Attention Network, and therefore are claimed essential to the validity of top-level conclusion.

3. Methodology

In the original paper, Residual Attention Network is constructed by stacking multiple Attention Modules. Each Attention Module is divided into two branches: mask branch and trunk branch. [1] The trunk branch performs feature processing and consists of multiple basic units. The mask branch outputs a soft feature mask which is applied to the output of the trunk branch to perform feature highlighting and is also composed of large numbers of basic units. The basic unit, as has been repeatedly referred from above, can be seen as a CNN block which outputs the sum of residual mapping of input and identity mapping of input. Depending on different residual mappings, the basic units can be changed to various versions. So in essence, Residual Attention Network can be seen as a network created by intelligently stacking and organizing a large number of the same basic units together.

In our work, we first replicate the bottleneck style Residual Unit from ResNet [4] as our basic unit to construct the trunk branch and mask branch. Then we use these branches to form the attention module. The attention residual learning module is also introduced and applied to the attention module here as a means to aggregate the output of trunk branch and mask branch. The attention modules are then stacked to create our base network. This architecture we followed is very similar to that of the original paper, as we believe the network built this way enables us to easily swap out certain modules with its varied version to test out the performance difference.

3.1 Objectives and Technical Challenges

The challenges we face in studying Residual Attention Network can be categorized into design challenges during network recreation phase and interpretation challenges during network experimentation phase. In this section, we will mainly focus on describing the challenge themselves and will talk about how we solve each of them in the implementation section.

Design Challenges

The first design challenge comes from the ImageNet datasets used in the paper. It consists of 1,000 classes with 1.2 million training images, 50,000 validation images, and 100,000 test images. The size of this dataset has well passed our available computational and storage resources limitations, so we decide to conduct our network training on the much smaller CIFAR-10 and CIFAR-100 datasets.

As we shift our recreation objective towards CIFAR dataset series, we face our second challenge. In each of the

Layer	Output Size	Attention-56	Attention-92
Conv1	112×112	7 × 7, 64, stride 2	
Max pooling	56×56	3 × 3 stride 2	
Residual Unit	56×56	$\begin{pmatrix}1\times1,64\\3\times3,64\\1\times1,256\end{pmatrix}\times1$	
Attention Module	56×56	Attention ×1	Attention ×1
Residual Unit	28×28	$\begin{pmatrix}1\times1,128\\3\times3,128\\1\times1,512\end{pmatrix}\times1$	
Attention Module	28×28	Attention ×1	Attention ×2
Residual Unit	14×14	$\begin{pmatrix}1\times1,256\\3\times3,256\\1\times1,1024\end{pmatrix}\times1$	
Attention Module	14×14	Attention ×1	Attention ×3
Residual Unit	7×7	$\begin{pmatrix}1\times1,512\\3\times3,512\\1\times1,2048\end{pmatrix}\times3$	
Average pooling	1×1	7 × 7 stride 1	
FC,Softmax	1000		
params×10^6		31.9	51.3
FLOPs×10^9		6.2	10.4
Trunk depth		56	92

Table 1: Residual Attention Network architecture details for ImageNet from the original paper. The items marked in red are subjects we mention in the challenge section.

ResNet basic units in our network, one of the parameters is a list of filter values required by the residual mapping. In the original paper, however, these filter parameter settings are only provided in detail for the network designed for ImageNet dataset, and when we construct our CIFAR designated network using these filter settings, we experience slow training time and signs of overfitting, with on average CIFAR-10 near 0% training error and stagnated 43% validation error and CIFAR-100 near 0% training error and stagnated 65% validation error.

After we realize the ill-fitted nature of the filter parameters setting for the basic unit, we decide to test out the validity of a few other parameters. In one of our tests, we find that the initial convolutional layer before the first attention module has filter size of 7 and stride size of 2 and is then followed by a maxpool layer with size of 3 and strides of 2. Since the size of image from the ImageNet is 224 by 224, the output after these operations would be 56 by 56, which is still large enough for the following convolutional operations. However, in our case of using CIFAR dataset, with original image size being 32 by 32, the output becomes as small as 8 by 8, which we believe is too small of an early-stage output to contain enough information for the following convolutional operations. This discovery makes us believe even more that the structure design proposed by the paper is solely meant for ImageNet. With this in mind, we deduce that the overly complex issue of our network could really come from any building components that are not firmly specified in the structure design. For instance, the number of stages, which is the number of attention modules stacked together, could also be the source of overfitting.

After the overfitting issue is addressed, we face the last challenge in our network design phase, and that is the performance optimization. Despite our attempts in trying various filter parameters combinations, our overall Top-1 error of the network stagnates at 38% on average for CIFAR-10 with 20 epochs. During this experimenting process, we discover one other topic that the paper failed to address in precision. The skip connection, which is a feature in the mask branch, is only briefly mentioned twice in the main body of the paper and its usage methodology is overlooked.

Interpretation Challenges

In order to conduct the experiments of testing the effectiveness of attention modules on network performance, we expect to create varied versions of the network with different numbers of attention modules in each stage. However, we find conflicting descriptions on how to design such alternative networks in the paper. In section 4.1 attention residual learning subsection, the paper states that "We set the number of Attention Modules in each stage m = {1, 2, 3, 4}. For Attention Module, this leads to Attention-56 (named by trunk layer depth), Attention-92, Attention-128 and Attention-164 respectively." [1] This implies that for the attention-92 network, the number of attention modules in each stage should be 2. However, in Table 2 of the original paper, which is displayed as Table 1 in our paper, the design description displays the attention-92 network has the numbers of attention modules being 1, 2, and 3 for each of the 3 stages, respectively.

Additionally, we hope to create each varied version of the network following the design description as closely as possible, so it is essential for us to understand the logic behind each module structure. With this in mind, we try to understand the naming convention of these varied attention network series, hence how each number in the network name is calculated. In section 4.1 implementation subsection, the original paper describes the number in network name as "The number of weighted layers in trunk branch", and it is "36m+20 where m is the number of Attention Module in one stage." [1] With the information given in Figure 2 of the paper, we are able to deduce the depth 20 in the formula comes from the extra 2 residual units in each stage. However, we cannot reverse engineer the depth 36 from the attention module using the design information given.

Lastly, the design description of the network proposed by the paper displays behaviors that cannot be explained by the structure of the unit being used. The Table 2 from the original paper, Table 1 in our paper, shows that one Residual Unit is applied before each of the attention modules. In addition, after the final attention module, three residual units are applied. Despite all being named the same, these residual units behave differently according to this table. The first residual unit generates an output (56x56) that is the same size as that of the input (56x56). However, for the rest of the residual units, they all have the effect of shrinking output size by half compared to input size. This shrinking behavior also happens for the three-residual-units stack. The paper does not mention the cause of these varied behaviors.

3.2 Problem Formulation and Design Description

The implementation of the original papers can be summarized into two phases: the network recreation phase and network implementation phase. In this section, we talk about the formulation of each phase in detail.

Network Recreation

In order to achieve high efficiency and convenience for network executing and testing, we decide to implement the design style where we would first divide the network into 5 blocks based on their functionalities, and then start to build each block using even smaller building units, in our case the basic units.

Residual Unit. In our base network, we use bottleneck style Residual Unit from ResNet [4] as our basic unit. The Batch Normalization is repeated three times in residual mapping to allow less dependency between each layer, therefore encouraging higher learning rates and mitigating overfitting issues. In addition, in order to address the challenge, we identified before, where residual units behave differently throughout network proposed by the paper, we introduce a new type of Residual Units: Residual Unit-Downsampling. Compared to the regular Residual Unit, a stride of 2x2 is introduced to the intermediate convolutional layer instead of 1x1 to achieve the halving effect described by the proposed design description. The Residual Unit-Downsampling is only applied to the network where the halving effect is identified. Unless specified, all Residual Units mentioned from now on refer to the regular Residual Unit.

Trunk Branch. Trunk Branch performs feature processing and is solely composed of a stack of Residual Units. the number of Residual Units stacked, t, is the hyperparameter. We use t=2 throughout our entire network.

Soft Mask Branch. Soft Mask Branch takes in the same input as Trunk Branch and extracts the location information of certain features with the bottom-up top-down structure. The bottom-up structure is achieved by applying max pooling several times with r number of residual units in between. After this, 2*r number of residual units are applied to perform feature identification. The top-down structure, on the other hand, uses Upsampling layers with r number of residual units in between to expand the feature information. The number of Upsampling layers is the same as that of max pooling to keep the output size the same as the input feature map. The skip connection is added between bottom-up and top-down structure to ensure sufficient gradient backflow and avoid vanishing gradient problems. Finally, two layers of 1x1 convolution and sigmoid activation are applied before the output layer. We use r=1 in our network.

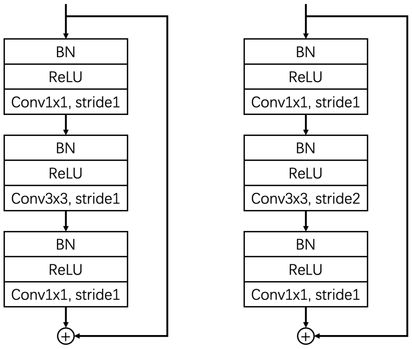


Figure 2: Left: The original Residual Unit design structure proposed by the paper. Right: The Residual Unit-Downsampling design structure we introduce to achieve the dimension halving effect.

Attention Residual Learning. Attention Residual Learning block acts as a means to aggregate the output of Trunk Branch and Soft Mask Branch. Its overall concept is similar to the residual learning introduced by ResNet. [4] With Mask output M(x) and Trunk output F(x), the output aggregation H(x) is the dot product of M(x) and F(x) plus M(x). This way, as described by the paper, would solve the feature degradation problem caused by simple dot production in deep layers.

$$H_{i,c}(x) = (1 + M_{i,c}(x)) * F_{i,c}(x)$$

Attention Module and Stage. With Trunk Branch, Soft Mask Branch, and Attention Residual Learning standing by, an Attention Module is simply created by putting them together, with one input being fed into both branches and their outputs being aggregated through ARL. A Stage is constructed by adding p number of Residual Unit before and after attention module. we use p=1 in our network.

Network Experimentation

In this section, we describe the design of the additional building blocks that are created for our network experimentation phase.

Local Mask Branch. Local Mask branch is a naive version of the Soft Mask Branch. It excludes the bottom-up top-down structure and only consists of 2 Residual Units. By comparing it with Soft Mask Branch, we would be able to test out the effectiveness of the bottom-up top-down structure.

Naive Attention Learning. In order to test out the effectiveness of Attention Residual Learning stated in the paper, we would need to create an alternative aggregation method to compare it with. Based on the definition of ARL, one reasonable method would be Naive Attention Learning, which is the simple dot production between Mask output M(x) and Trunk output F(x).

Attention-56/92. The authors propose different network architectures in their paper, namely Attention-56, Attention-92, and Attention-128. The key difference between them is the underlying number of attention modules implemented in a stage. For instance, in the Attention-56 network, the number of attention modules in each stage m is 1. The number in the network's name is calculated as 36m + 20, which is stated as the Trunk layer depth by the paper. As for the conflicting information about the architecture design of Attention-92 stated in our challenge section, we decide to comply solely with the description stated in the writing of the paper.

4. Implementation

In this section we talk about the implementation detail of our network recreation and experimentation. This includes the detailed description of the datasets we use, the data preprocessing we perform, the training process we implement, and the guideline we follow in conducting our network experiments.

Dataset and Data preprocess

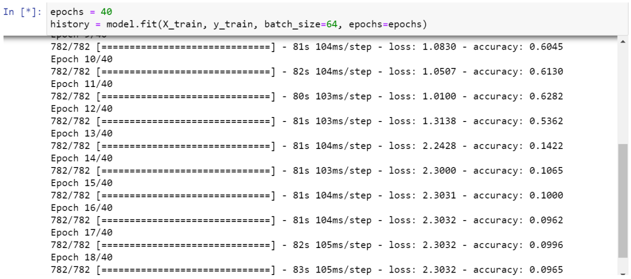
As mentioned before, we use CIFAR-10 and CIFAR-100 datasets in our study. The CIFAR-10 and CIFAR-100 datasets consist of 60,000 32 × 32 color images of 10 and 100 classes respectively, with 50,000 training images and 10,000 test images. For each image class, CIFAR-10 dataset consists of 6000 images whereas CIFAR-100 consists of 600 images.

With aim to achieve better network generalization, we perform data augmentation to our dataset by utilizing the ImageDataGenerator API from Keras. In this process, random selected images are rotated, shifted, and flipped and add back to the original dataset. We then divide each image array by 255 to normalize the input to [0, 1] range. From experimenting, we also found that by random shuffling the training data before feeding it into the model, we are able to achieve higher network performance. Therefore, this shuffling step is also added into our data preprocessing.

Network Training

In our attempt of training and optimizing the network, we focus on key variation factors such as optimization methods, batch sizes, and the network parameter settings. We first compare the performance between Stochastic Gradient Descent (SGD) and Adam Optimizer. We then discuss how batch size influences the training process. Finally, we analyze how different parameters affect the model accuracy and generalization.

Optimization Methods. We initially choose Adam as our optimizer. Although being a popular optimization method in Deep Learning, in our case, Adam exhibits sizable instability. In some of our instances, the accuracy is able to boost up to 60% within the first 10 epochs, but it then drops dramatically before slowly climbing back up again. We suspect the reasons behind this phenomenon could be the unsuitable magnitude of learning rate convergence to different local minima. We try to solve this issue by initializing different learning rates and epsilons, but the problem is still present sometimes.



```
In [*]: epochs = 40
        history = model.fit(X_train, y_train, batch_size=64, epochs=epochs)

        782/782 [==============================] - 81s 104ms/step - loss: 1.0830 - accuracy: 0.6045
        Epoch 10/40
        782/782 [==============================] - 82s 104ms/step - loss: 1.0507 - accuracy: 0.6130
        Epoch 11/40
        782/782 [==============================] - 80s 103ms/step - loss: 1.0100 - accuracy: 0.6282
        Epoch 12/40
        782/782 [==============================] - 81s 103ms/step - loss: 1.3138 - accuracy: 0.5362
        Epoch 13/40
        782/782 [==============================] - 81s 104ms/step - loss: 2.2428 - accuracy: 0.1422
        Epoch 14/40
        782/782 [==============================] - 81s 103ms/step - loss: 2.3000 - accuracy: 0.1065
        Epoch 15/40
        782/782 [==============================] - 81s 104ms/step - loss: 2.3031 - accuracy: 0.1000
        Epoch 16/40
        782/782 [==============================] - 81s 104ms/step - loss: 2.3032 - accuracy: 0.0962
        Epoch 17/40
        782/782 [==============================] - 82s 105ms/step - loss: 2.3032 - accuracy: 0.0996
        Epoch 18/40
        782/782 [==============================] - 83s 105ms/step - loss: 2.3032 - accuracy: 0.0965
```

Figure 3: Training instance under Adam optimizer when training accuracy experience sudden drop.

Instead, we turn our attention to Nesterov SGD optimizer. We set the learning rate to 0.1 and use a decay rate of 0.0001 and momentum of 0.9. The learning rate is divided by 10 at 80th epoch and 120th epoch using Keras LearningRateScheduler API. Compared to Adam, accuracy increases slower but in a more stable fashion. Furthermore, the training loss using Nesterov SGD optimizer is not as good as that of using Adam optimizer, but the validation loss is lower.

Batch Size. Batch size affects overall training time and convergence of the optimizer. The goal is to find an appropriate batch size, where the optimizer achieves good convergence meanwhile the training process doesn't take too long. A good choice of batch size often varies from 32 to 512, usually a power of 2, depending on the dataset and hardware. Initially, we set the batch size to 256. This setting results in fast convergence on training data, however, the network does not generalize well on testing data. We believe this is an indication of batch size being too large. Therefore, different batch sizes are tested, and based on the results we conclude that 32 and 64 are appropriate batch sizes for training our network as they result in much better generalization.

Network Parameters Setting. Filter size, kernel size, strides, Max Pooling layers are subjects under test within our network parameter settings. We display the final design description of these parameters in Table 2. To address the initial overfitting issue, we decide to use the quartered version of the filter parameter settings proposed by the original paper. This method solves the overfitting issue. In addition, to address the challenge we face regarding to skip connection features in Soft Mask Branch, we decide to implement the design where we would first have 2 skip connections in the first stage and subtract 1 skip connection for each of the following stages. This approach is the same as the one proposed by the paper, but we believe the number of skip connections in each stage does not have to follow this decreasing pattern and can well be any integer numbers between 0 and 2.

Network Experimentation

Given limited computational resources, we use CIFAR-10 dataset and 100 epochs to conduct all our experiments. More importantly, to ensure fairness and comparability of each pair of experiment results, we keep all other structures of the network and training methods the same except for the components being tested.

Layer	Output Size	Attention-56	Attention-92
Conv1	16 × 16	3 × 3, 64, stride 2	
Residual Unit	16 × 16	$\begin{pmatrix}1\times1,16\\3\times3,16\\1\times1,64\end{pmatrix}\times1$	
Attention Module - A	16 × 16	Attention × 1	Attention × 2
Residual Unit Down Sampling	8 × 8	$\begin{pmatrix}1\times1,32\\3\times3,32\\1\times1,128\end{pmatrix}\times1$	
Attention Module - B	8 × 8	Attention × 1	Attention × 2
Residual Unit Down Sampling	4 × 4	$\begin{pmatrix}1\times1,64\\3\times3,64\\1\times1,256\end{pmatrix}\times1$	
Attention Module - C	4 × 4	Attention × 1	Attention × 2
Residual Unit Down Sampling	2 × 2	$\begin{pmatrix}1\times1,128\\3\times3,128\\1\times1,512\end{pmatrix}\times1$	
Residual Unit	2 × 2	$\begin{pmatrix}1\times1,128\\3\times3,128\\1\times1,512\end{pmatrix}\times2$	
Average Pooling	1 × 1	2 × 2 stride 1	
FCSoftmax	10		
params $\times10^6$		4.08	5.76
Trunk depth		56	92

Table 2: Residual Attention Network architecture details for CIFAR-10.

```
Network 1 Residual Attention Network(Attention 56)
Input: Image
Output: Labels
  Processing:
    1.ImageDataGenerator
      1.1 Rotate
      1.2 Shift
      1.3 Flip
    2.Normalize
  end Processing:
Preparation Phase
Input: Image Tensor                                  ▷ Shape: 32 × 32
  Processing:                                        ▷ Output Size
    1. Padding                                       ▷ 40 × 40
    2. Cropping                                      ▷ 32 × 32
    3. Convolution                                   ▷ 16 × 16
    4. MaxPooling                                    ▷ optional
  end Processing:
Stage 1
Input: Image Tensor, Filter, residual unit type
  Processing:                                        ▷ Output Size
    1. Residual Unit                                 ▷ 16 × 16
    2. Attention Module A * 1                        ▷ 16 × 16
  end Processing:
Stage 2
Input: Image Tensor, Filter, residual unit type
  Processing:                                        ▷ Output Size
    1. Residual Unit-Downsampling                    ▷ 8 × 8
    2. Attention Module B * 1                        ▷ 8 × 8
  end Processing:
Stage 3
Input: Image Tensor, Filter, residual unit type
  Processing:                                        ▷ Output Size
    1. Residual Unit-Downsampling                    ▷ 4 × 4
    2. Attention Module C * 1                        ▷ 4 × 4
  end Processing:
Ending Phase
Input: Image Tensor, Filter, residual unit type
  Processing:                                        ▷ Output Size
    1. Residual Unit-Downsampling                    ▷ 2 × 2
    2. Residual Unit * 2                             ▷ 2 × 2
    2. Average Pooling                               ▷ 1 × 1
    3. Fully Connected
  end Processing:                                    Output Size
```

Figure 4: Attention-56 Pseudo code detail for CIFAR-10.

5. Results

5.1 Project Results

Using the configurations we describe above, our best Residual Attention Network achieves 14.43% test Top-1 error on CIFAR-10 dataset and 48.93% test Top-1 error on CIFAR-100 dataset. As for the result of our experiments, we are able to show that the network performance using Soft Mask Branch is superior to that of using Local Mask Branch. Moreover, our testing result on Attention Residual Learning vs. Naive Residual Learning illustrates Attention Residual Learning provides performance benefit to the network as well.

Network	CIFAR-10 Top-1 err. %	CIFAR-100 Top-1 err. %
Attention-56	14.43	48.93
Attention-92	22.93	45.8

Network	ARL Top-1 err. %	NAL Top-1 err. %
Attention-56	14.43	15.02

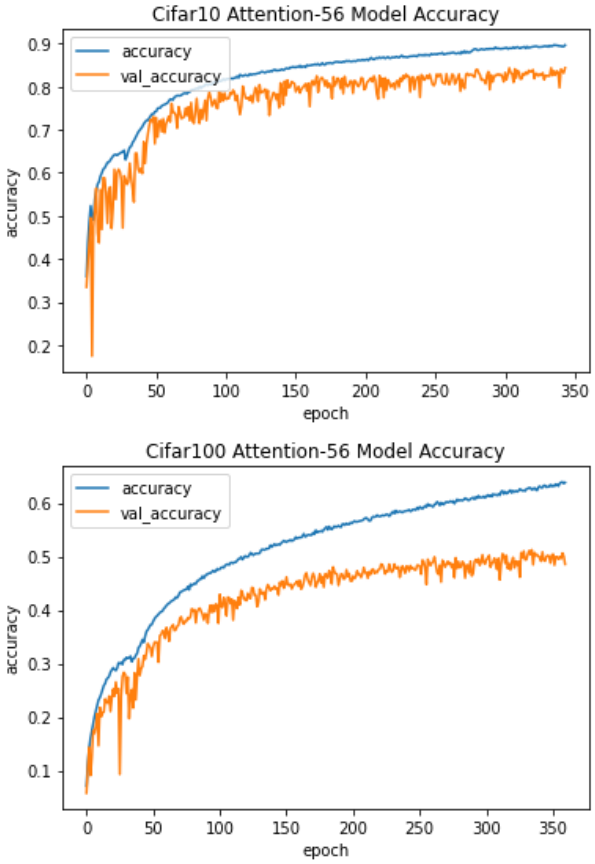


Figure 5: CIFAR-10 and CIFAR-100 runtime accuracy. The Top-1 error results are calculated as 1 minus accuracy.

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

Although our network performance does not match the competitive performance stated by the original paper, our Attention-56 network only uses 4.08x 10^6 number of parameters as compared to 31.9x10^6 parameters from the network in the paper. This significant size difference in parameter size means our network is 8 times lighter in terms of network weight, which would guarantee a much faster training time. In terms of the experiments result, we are able to confirm the result from the paper that ARL and Soft Mask offer performance improvement to the network. However, for Attention-92 network, we fail to confirm its superior performance over Attention-56, as our result show potential degradation problem of the Attention-92.

Due to the fact that the code for the trained network described by the paper is not publicly available, we are unable to compare our training/validation error per training time with that of the paper.

5.3 Discussion of Insights Gained

We learn from experience that patience is the key during training procedure, especially for deep networks. During the first few days of training our network, when we stop seeing accuracy increasing significantly for several epochs in a row, we would deem the design structure being not effective enough and interrupt the runtime. But it turns out that the training procedure for deep networks like Residual Attention Network could usually take up to hundreds of epochs to gradually achieve the desired performance.

In our opinion, the paper lacks sufficient descriptions of network design for CIFAR series datasets. This creates many loose ends and subjects for viewers' own explorations and interpretations. The challenges we face in our recreation and experimentation phases are among those subjects. Due to this nature, the difference in the performance result could come from many aspects. For instance, whether the data augmentation and random shuffling are performed, how the halving effect of the residual unit is actually achieved, the actual filter setting for CIFAR datasets, the actual number of stages implemented for CIFAR datasets, etc.

Network	CIFAR-10 (Top-1 err. %)
Attention-56	14.43
Naïve Network	22.08

In addition, when we try to improve the network performance on CIFAR-10 dataset, we also become curious of the maximum value-add of this type of very deep network. So as a quick experimentation, we create a simple convolutional network with 8 layers, which is about 1/40 the depth of the Attention-56 network. It turns out this naive network is able to achieve as low as 22.08% of top 1 error, which is much lower than what we expect. With our trained model having 14.43% top 1 error, we can say that the maximum value-add of this deep network, compared to a naive everyone-can-do type network, is 7.65% top 1 error rate. We believe this little exercise is a good way to demonstrate the real challenge deep learning is trying to tackle, which in some sense can be described as the diminishing performance margin gain. In other words, as the performance of solving a task with a model gets better, it will take a new model more effort and computational power to achieve the same unit of performance increase. However, this can also demonstrate the cost-effectiveness of deep networks and remind us that going deep is not the solve-all solution, and we should always be mindful with our model design and consider the type and final goal of tasks at hand.

6. Conclusion

We aim to study and understand the Residual Attention Network proposed by the original paper. We approach our study by recreating the network and conducting experiments to test the effectiveness of its building components. The results from our experiments enable us to validate the lower-level conclusion drawn by the original paper. The performances of our network, with xxx top 1 error for CIFAR-10 and xxx top 1 error CIFAR-100, however, cannot match the result stated in the original paper. In the future, we will explore the feasibility and performance variation of training CIFAR designated Residual Attention Network on the dataset with identical structure but different image contents and classification labels.

7. Acknowledgement

We would like to offer my special thanks to our colleague Jiana Feng for generously sharing some design insights that allow us to achieve higher network performance. We would also like to express our appreciation to Professor Zoran Kostic for giving us more precise guidelines on the structure of our research paper.

References

[1] Fei Wang, Mengqing Jiang, Chen Qian, Shuo Yang, Cheng Li, Honggang Zhang, Xiaogang Wang, and Xiaoou Tang Residual Attention Network for Image Classification. arXiv:1704.06904, 2017, pp. 1.

[2] K. He, X. Zhang, S. Ren, and J. Sun. Identity mappings in deep residual networks. arXiv preprint arXiv:1603.05027, 2016, pp. 1.

[3] S. Zagoruyko and N. Komodakis. Wide residual networks. arXiv preprint arXiv:1605.07146, 2016, pp. 1.

[4] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. arXiv:1512.03385, 2015, pp. 1.

[5] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural Machine Translation by Jointly Learning to Align and Translate. arXiv:1512.03385, 2015, pp. 1.

[6] https://buomsoo-kim.github.io/attention/2020/01/01/Attention-mechanism-1.md/

Appendix

Individual Student Contributions in Fractions

	HT2459	JW3864
Last Name	Tian	Wang
Fraction of (useful) total contribution	1/2	1/2
What I did 1	Network tuning and optimization	First draft network creation
What I did 2	Paper writing	Paper writing
What I did 3	Daily discussion on the project	Daily discussion on the project