RescaleViT: Training Vision Transformer without Normalization

Tao Chaofan Deng Weipeng
Department of Electrical and Electronic Engineering
the University of Hong Kong
cftao@connect.hku.hk
dengf330@connect.hku.hk

Abstract

Normalization is an important technique in training deep neural network, which helps to avoid gradient exploding/vanishing and internal covariate shift problems. However, the theory behind why normalization works is still unclear. Some researchers wonder if they can train deep neural network without normalization and can still obtain the acceptable performance. In this paper, we proposed RescaleViT, it is a transformer model which without normalization and achieves the comparable performance with training speedup on different variants of vision transformer (ViT). Codes are available[1].

1 Introduction

In recent years, deep neural network has achieved good results in many different applications, such as image recognition[13, 5, 6, 15], object detection[10, 4] and natural language processing[16, 2, 14]. However, as network structure becomes deeper, training becomes more difficult. Deep neural network with many hidden layer has some problems, such as gradient explosion or gradient vanishing which directly makes the neural network training fail. Various normalization techniques can help mitigate this problem to some extent. Batch Normalization (BN)[7] can solve this problem by control the variance and distribution between each hidden layer. Although it is mainly to solve the problem of Internal Covariate Shift, BN can also solve the gradient vanishing or exploding problems by pulling the mean value of each hidden layer back to 0 and the variance back to 1 as well. There are many normalization variation for different types of tasks: Layer Normalization (GN)[17] is using in mini-batch training, Layer Normalization (LN)[1] is usually using in recurrent neural network (RNN). Now, almost training any deep neural network uses normalization techniques.

Although normalization has proved to be effective in many different network structures, there is no perfect mathematical proof of why it works. Batch normalization was originally proposed to solve the internal covariate shift problem[7] and it is also makes training deep neural network easier. Because the reason why normalization is effective is not clear and the normalization has been found to can help training neural network with deep structures during experiment. Some scholars have asked a question: Is normalization essential in training deep neural network? Some new techniques have been proposed to achieve the same effect as normalization. RescaleNet is a modify ResNet network structure proposed by Hu Kai which can work without normalization layer and still achieve the same or slightly better result.[11]

Transformer[16] is a powerful attention-based Encoder-Decoder model which was originally proposed by Google to do machine translation. It is a entirely attention-based Seq2Seq[14] architecture and without any RNN part. Therefore, it can support parallelization.In addition, attention mechanisms can also capture remote information.

[1] https://github.com/ChaofanTao/RescaleViT

In this paper, we proposed a new network Rescale Vision Transformer (RescaleViT) which does not require normalization. RescaleViT is based on ViT[3] a transformer architecture for vision.

We validate our method on CIFAR-10 and CIFAR-100[8] datasets and compare with different version of ViT which with layer normalization, batch normalization and group normalization. Our proposed RescaleViT can achieve the same or slightly better performance and improved the training speed.

2 Related Work

2.1 Normalization

Normalization is one of the most important and frequently used techniques in training deep neural network. Batch Normalization (BN) solve the gradient exploding or gradient vanishing problems by drag the data distribution between layers back to with mean to 0 and variance to 1. BN enable training with a large gradient so that the training process can be faster. Layer Normalization (LN) is the normalization of all the inputs to the neurons which within a same layer in the neural network, and it can reduce the computation cost of normalization in RNN. Group Normalization (GN) separate channels into groups and compute normalization over each groups, and it can reduce the normalization computation cost and more effective for small batch setting.

2.2 Transformer

2.2.1 Network Structure

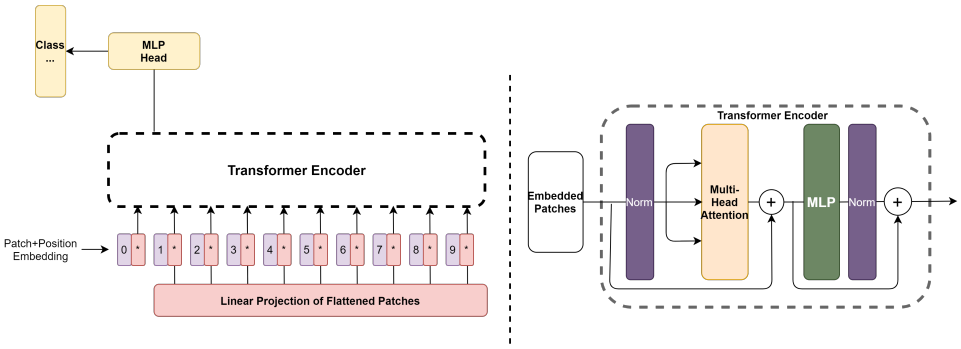


Figure 1: Structure of ViT. the input images will be split into fix-size patches, these patches will be linearly embed and add position embeddings. The result of the linear projection layer will be feed into a standard Transformer encoder. The standard encoder will take the input pass a normalization layer first. The next layer is multi-head attention layer and there is a residual connection here. $Input_i^{MLP} = Input_i + Attention(Input_i)$, which $Input_i$ is the input of the i-th encoder block. The final output of this encoder block can be written as $output_i = Input_i^{MLP} + Norm(MLP(Input_i^{MLP}))$.

Transformer completely abandons the CNN or RNN. The attention based mechanism have better long-distance memory ability and support parallel computing naturally. Once it was proposed, it achieved top results in many different natural language processing tasks and it has attracted the attention of researchers from computer vision. Vision Transformer[3] (ViT) is a variant of transformer in computer vision and it is entirely based on attention without any CNN parts. ViT can achieve results that are competitive with the current optimal vision model by per-trained on large data sets and transfer to standard data sets. Besides, ViT requires significantly less computing resources.

ViT has almost the same network structure with the originally transformer proposed by Google. It's structure is shown in Fig.1. Let $Input_0$ be the input of the network and $Input_i$ represent the input of the i-th encoder block. The input of $i+1$-th encoder block $Input_{i+1}$ is the output of the i-th encoder clock $output_i$. So the formula can be written as $output_i = E(Input_i)$ which $E()$ represents the encoder block. $Input_0$ will pass an input embedding and positional encoding block to transform to the input of the first encoder $Input_1$. Then each encoder block contains two main part: Attention part and Feed Forward part. There are also two normalization layer which locate after the attention layer and the feed forward layer. Let $x_{i,a}$ represents the output of the attention layer and $Norm(x)$

represents do the Normalization operation on x. The input of feed forward layer $Input_i^{MLP}$ can be written as

$$Input_i^{MLP} = Input_i + Attention(Norm(Input_i)) \tag{1}$$

There is a residual connection here. After the first normalization layer the output $Input_i^{MLP}$ will pass through a feed forward layer which also along with a normalization layer after that. The output of the second normalization layer in this encoder block can be written as:

$$output_i = Input_i^{MLP} + Norm(MLP(Input_i^{MLP})) \tag{2}$$

$MLP()$ represents the feed forward layer. In general, feed forward layer contain a multi-layer fully connected neural network.

The overall structure of decoder block is almost the same as the encoder block. The decoder also contains attention layers, feed forward layer and normalization layers. The decoder gets two input, one is to use the output of the decoder as the residual input and the other is the begin token if the transformer is doing NLP tasks.

2.2.2 Attention

The most important component in transformer is attention mechanism and the authors use an attention called self-attention. The formula given below:

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V \tag{3}$$

Q is query matrix, K is keys and V represents values. As figure.2 shows a^i represents the i-th input token or the i-th output token from the previous layer. Then we can get 3 different matrix from each a^i by conducting the following calculation:

$$q : query : q^i = W^q a^i \tag{4}$$

$$k : key : k^i = W^k a^i \tag{5}$$

$$v : informationtobeextracted : v^i = W^v a^i \tag{6}$$

W^q, W^k, W^v are three different weights. Each a^i will multiple by these three matrix and get corresponding q^i, k^i, v^i matrix. Let's using the left part of figure.2 as example. For calculating the output information from a^1, self-attention mechanism will use the query q^1 of a^1 to multiple with all the keys including k^1 itself. The output of this process can be written as $a_{i,j}$ which i means the i-th query and j means the j-th key. So in the example, $a_{1,j}$ can be calculated by:

$$a_{1,1} = q^1 k^1$$
$$a_{1,2} = q^1 k^2$$
$$a_{1,3} = q^1 k^3$$
$$a_{1,4} = q^1 k^4$$

$a_{1,1}$ to $a_{1,4}$ can be consider as the relevant score between a^1 and others. So we get 4 attention scores in this example. These 4 attention scores will pass through a softmax layer and get $a'_{1,1}$ to $a'_{1,4}$. Finally, the output can be calculate by:

$$b^1 = a'_{1,1} v^1 + a'_{1,2} v^2 + a'_{1,3} v^3 + a'_{1,4} v^4 \tag{7}$$

Self-attention can extract information from long-distance based on the attention scores. Actually, most current transformer implementations are used another attention version called Multi-head attention.

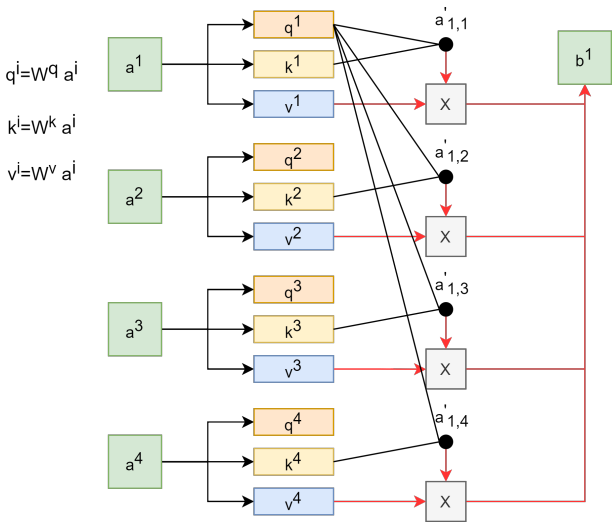


Figure 2: In attention mechanism, there are 3 weight matrix: W^q, W^k, W^v which refer to queries, keys, values, respectively. For example, each input patches $a^1, a^2, ..., a^4$ will multiple by these three weight matrix and get q^i, k^i, v^i. In order to calculate output, each query will multiple all the keys and the multiple results are the attention scores.For example calculate the $a^{'}_{1,j}$ which 1 represents the first query and j represents the j-th key, the score can be written as $a^{'}_{1,j} = q^1 k^j$. The final output b^1 is calculated by the weighted sum of v^i: $b^1 = a^{'}_{1,1} v^1 + a^{'}_{1,2} v^2 + a^{'}_{1,3} v^3 + a^{'}_{1,4} v^4$.

3 Preliminaries

We first introduce the key problems in training deep neural network.

3.1 Gradient explosion and vanishing

The main issue needed to be solved in training very deep neural network is the gradient explosion or vanishing problem. This is because the accumulation of gradients in back propagation. To explain this problem, let's consider a fully connect neural network with 3 hidden layer as example which shown in figure.3. In gradient descend, weight is updated by:

$$w_i = w_i - \frac{dL}{dw_i} \tag{8}$$

L is the loss function and w_i is the weight which will be updated. If the weight of first layer is going to be updated, based on chain rule of derivative, the $\frac{dL}{dw_1}$ can be calculate by:

$$\frac{dL}{dw_1} = \frac{dL}{df_4}\frac{df_4}{df_3}\frac{df_3}{df_2}\frac{df_2}{df_1}\frac{df_1}{dw_1} \tag{9}$$

f_i is the output of i-th layer. The formula can be re-written as :

$$\frac{dL}{dw_1} = \frac{dL}{df_4} f_4^{'} f_3^{'} f_2^{'} f_1^{'} w_4 w_3 w_2 \tag{10}$$

The derivative will continuous multiple in the back propagation process. If the gradient is large, then it will get very large gradient after many multiplication, this called gradient exploding. If the gradient is small than 1, after multiplication the final gradient will very close to 0. This call gradient vanishing.

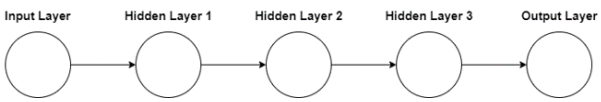


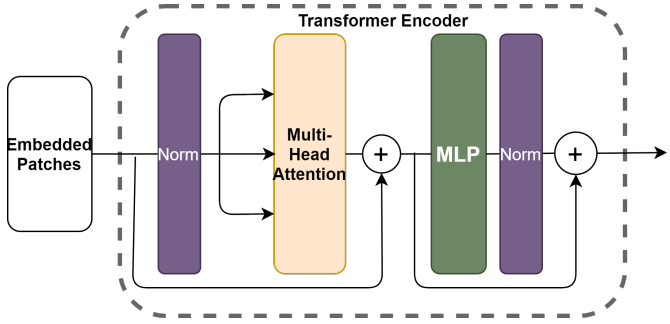
Figure 3: Example for gradient exploding or vanishing. A three hidden layers neural network, let f_i represents the function of i-th layer. The output of i-th layer or the input of $i+1$-th layer can be written as $x_{i+1} = f_i(x_i)$. $f_i = w_i x_{i-1} + b_i$.

3.2 Internal Covariate Shift

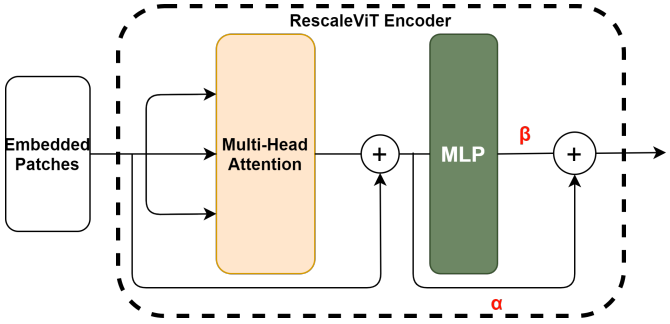
There is an important assumption in machine learning. If the model wants to achieve good performance, the training data and the testing data should follow the independent identically distributed. However, in deep neural network, each layer can be consider as taking the output form the previous layer as input and the output goes to the next layer. During the process, the hidden layer can change the distribution of the data. Therefore, the data distribution in the hidden layer is not stable, which will have a bad impact on learning.

4 RescaleViT

In this section, we will present RescaleVit which is a version of vision transformer that doesn't require normalization. Figure.4 shows the network structure of our RescaleViT. Instead of using



(a) ViT



(b) RescaleViT

Figure 4: Structures of RescaleViT and ViT. Our proposed RescaleViT removes the normalization layer and introduces two new hyper-parameters α and β which called rescaling. The rescaling process is conducting after the attention layer which can be written as $Output_i = \alpha_i Input_i^{MLP} + \beta_i MLP(Input_i^{MLP})$.

normalization layer, we introduce two new hyper-parameters α and β, recall the original ViT formula, we change it into:

$$Output_i = \alpha_i Input_i^{MLP} + \beta_i MLP(Input_i^{MLP}) \tag{11}$$

where $\alpha_i^2 + \beta_i^2 = 1$

The rescaling is conducted after the attention layer. The weights is initialized by He initialization[5], the inputs and outputs of feed forward $MLP()$ have approximately equal variance: $Var[MLP(Input_i^{MLP})] \approx Var[Input_i^{MLP}]$.

If the correlations between $output_i$ and $MLP(Input_i^{MLP})$ are small.[18] we can get:

$$Var(output_i) \approx Var(\alpha_i Input_i^{MLP}) + Var(\beta_i MLP(Input_i^{MLP})) = \alpha_i^2 Var(Input_i^{MLP}) + \beta_i^2 Var(MLP(Input_i^{MLP})))$$

Recall:$Var[MLP(Input_i^{MLP})] \approx Var[Input_i^{MLP}]$

$$\text{and:}\alpha_i^2 + \beta_i^2 = 1$$
$$\text{then:}Var(output_i) = Var(Input_i^{MLP})$$

$output_i$ is the input of next layer $input_{i+1}$. Let $F()$ represents the attention process in the first part of an encoder and it is a fix function, we can rewritten Eq.11 as:

$$Input_{i+1} = \alpha_i F(Input_i) + \beta_i MLP(F(Input_i)) \tag{12}$$

Eq.12 can be expanded as:

$$output_L = (\prod_{q=1}^{L} \alpha_q)F(Input_0) + \sum_{i=1}^{L} \beta_i \prod_{q=i+1}^{L} \alpha_q MLP(F(Input_{x_{i-1}})) \tag{13}$$

The optimal coefficients should ensure different blocks have the same weight:

$$\forall i \neq i^{'}, \beta_i \prod_{q=i+1}^{L} \alpha_q = \beta_{i'} \prod_{q=i'+1}^{L} \alpha_q \tag{14}$$

Solving Eq.14 with $\alpha_i^2 + \beta_i^2 = 1$,we get:

$$\alpha_i = \sqrt{\frac{i-1+c}{i+c}} \tag{15}$$

$$\beta_i = 1/\sqrt{i+c} \tag{16}$$

where c is a hyper-parameter, we choose $c = 1$ as our parameter, we can solve out:

$$\alpha_i = \sqrt{\frac{i}{i+1}} \tag{17}$$

$$\beta_i = \frac{1}{\sqrt{i+1}} \tag{18}$$

4.1 Image preprocessing

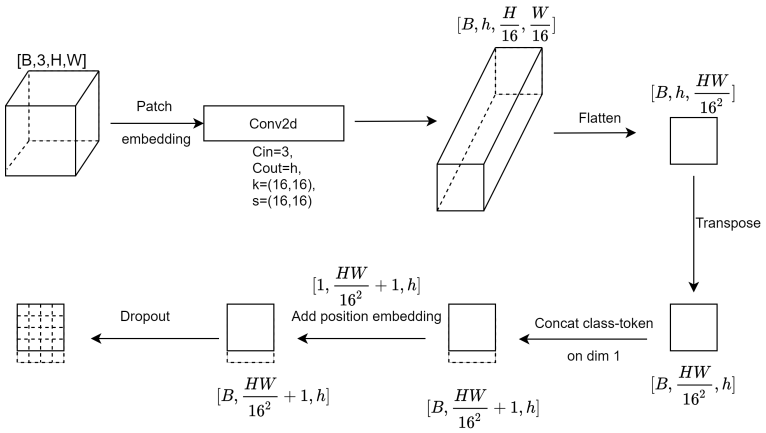


Figure 5: Mini-batch Image preprocessing for transformer usage. In ViT we use channel first notation. The input images will first go through a convolution layer to reduce dimensions on heigh and width, then the output will be flatten on the dimension $1st$ and $2rd$ and the output will be transposed, in order to get representation [Batch, length of signal, hidden dim]. Class tokens will be concatenated on dimension 1 and the whole matrix will add a position embedding which has the same dimension. The final output will go through dropout before sending into transformer.

As Fig.1 shows that the model takes images patches as input. However the image have to reduce dimensions before splitting, otherwise the dimensions will be too large. In this case the height and width

Table 1: ViT-B_16 result on CIFAR-10

model	LN	BN	GN	Rescale	resolution	acc(%)	time(min)
ViT-B_16	+				224x224	69.7	171.1
ViT-B_16*	+				224x224	98.4	170.6
ViT-B_16		+			224x224	66.5	181.1
ViT-B_16			+		224x224	65.1	185.1
ViT-B_16				+	224x224	65.2	165.9

Table 2: ViT-B_32 result on CIFAR-10

model	LN	BN	GN	Rescale	resolution	acc(%)	time(min)
ViT-B_32	+				224x224	67.6	68.5
ViT-B_32*	+				224x224	98.4	67.7
ViT-B_32		+			224x224	67.1	68.6
ViT-B_32			+		224x224	65.5	73.0
ViT-B_32				+	224x224	66.1	66.7

Table 3: ViT-L_16 result on CIFAR-10

model	LN	BN	GN	Rescale	resolution	acc(%)	time(min)
ViT-L_16	+				224x224	66.3	306.2
ViT-L_16*	+				224x224	99.0	308.2
ViT-L_16		+			224x224	63.0	326.4
ViT-L_16			+		224x224	60.8	330.9
ViT-L_16				+	224x224	62.9	295

Table 4: ViT-L_32 result on CIFAR-10

model	LN	BN	GN	Rescale	resolution	acc(%)	time(min)
ViT-L_32	+				224x224	67.5	146.2
ViT-L_32*	+				224x224	98.9	144.3
ViT-L_32		+			224x224	67.4	152.5
ViT-L_32			+		224x224	65.6	163.3
ViT-L_32				+	224x224	65.4	145.8

will be 224*224. A convolution core which with parameters: $Cin = 3, Cout = h, k = (16, 16), s = (16, 16)$ can reduce the height and width 16 times, so the dimensions become $[B, h, \frac{H}{16}, \frac{W}{16}]$ which h is the hidden feature number. The next step is flatten and transpose the second and third dimensions, the result becomes $[B, \frac{HW}{16^2}, h]$. Class tokens have to be concatenated on dimension 1 before feeding into transformer. The dimension becomes $[B, \frac{HW}{16^2} + 1, h]$. And we will add a position embedding matrix which has the same dimension. The last process is dropout and the final dimension is still $[B, \frac{HW}{16^2} + 1, h]$.

5 Experiments

In order to test our RescaleViT, we conduct some experiments on CIFAR-10 and CIFAR-100 data sets and we also compared our model with ViT. We conduct all single experiment on one NVIDIA-3090 card. The default batch size is 128. We train 20000 iterations for all experiments. All the evaluation is made every 100 iteration. We use float-point-16 training powered by Automatic Mixed Precision(Amp) [9], which reduces GPU memory usage 3 4x during training.

The table below shows the experiment result of 4 different types of vision transformer architecture: ViT-B_16, ViT-B_32, ViT-L_16 and ViT-L_32. "+" denotes the used normalization method or Rescale method. We use "*" denote the model initialized with pre-tained checkpoint on Imagenet21k. "16" and "32" represent the kernel setting of the convolution layer in image preprocessing. We train the model using fp16 precision.

5.1 Results

Our RescaleViT without normalization can achieve almost the same performance in every type architectures of vision transformer. In most types of ViT, models with layer normalization get the top places and our RescaleViT can perform better than using group normalization in most cases. Notably, our RescaleViT can significantly reduce the training time and reduces up to 35 minutes compared with group normalization in ViT-L_16.

In CIFAR-100 dataset, the experiment result looks pretty much the same. Our RescaleViT doesn't lose much in performance even without normalization and our model improves training speed.

From the results, we can observe that the pre-trained checkpoint on the ImageNet21k plays a significant role in training ViT, for various architectures. The model used pre-trained checkpoint outperform the CNN-based model.

On the other hand, for the model trained from scratch, the results cannot beat CNN-based model, which is consists with the observation in the paper ViT.

Besides the past experience about CNN, we have several empirical deductions.

1) Transformer has potential to outperform CNN-based model with a large margin, while it generally needs more data, more parameters and more training time.

2) Transformer has potential to outperform CNN-based model with a large margin, while it generally needs more data, more parameters and more training time.

Table 5: ViT-B_16 result on CIFAR-100

model	LN	BN	GN	Rescale	resolution	acc(%)	time(min)
ViT-B_16	+				224x224	43.5	168.8
ViT-B_16*	+				224x224	91.9	167.7
ViT-B_16		+			224x224	38.7	177.5
ViT-B_16			+		224x224	38.2	181.4
ViT-B_16				+	224x224	36.1	162.6

Table 6: ViT-B_32 result on CIFAR-100

model	LN	BN	GN	Rescale	resolution	acc(%)	time(min)
ViT-B_32	+				224x224	42.8	68.4
ViT-B_32*	+				224x224	91.0	65.6
ViT-B_32		+			224x224	39.9	68.7
ViT-B_32			+		224x224	40.0	73.1
ViT-B_32				+	224x224	37.8	67.5

Table 7: ViT-L_16 result on CIFAR-100

model	LN	BN	GN	Rescale	resolution	acc(%)	time(min)
ViT-L_16	+				224x224	40.2	303.9
ViT-L_16*	+				224x224	93.2	298.8
ViT-L_16		+			224x224	34.5	320.5
ViT-L_16			+		224x224	31.6	326.7
ViT-L_16				+	224x224	34.6	295.6

Table 8: ViT-L_32 result on CIFAR-100

model	LN	BN	GN	Rescale	resolution	acc(%)	time(min)
ViT-L_32	+				224x224	42.7	144.6
ViT-L_32*	+				224x224	92.2	143.4
ViT-L_32		+			224x224	40.6	150.2
ViT-L_32			+		224x224	39.6	160.5
ViT-L_32				+	224x224	37.6	143.6

3) Generally speaking, LN > BN = Rescale > GN on the CIFAR10 and CIFAR100 in the performance.

4) Both pre-trained model and RescaleViT can speedup the training time a little bit.

5.2 Attention Map

The ViT consists of a Standard Transformer Encoder, and the encoder consists of Self-Attention and MLP module. The attention map for the input image can be visualized through the attention score of self-attention.

Here is the learned attention map of ViT-B_16 on CIFAR-10 dataset.(Fig.6) We visualize the attention map at fisrt head in 0, 3rd, 6th ,9th block.

We can observe that the attention map is sparsely distributed regardless of normalization methods. It is noteworthy that the pre-trained LN-model learn the most important components along the diagonal, which means the learned attention is focus on each neuron itself. RescaleViT has more sparse attention than the model using normalization.

5.3 Loss and Accuracy

The validation loss and accuracy are illustrated below. Architecture: ViT-B_32, Dataset:CIFAR-10.

Green line is loss curve and blue line is accuracy curve. The model with (LN+pre-trained) has a good initialization that even achieve over 90% accuracy before training, and it converges quickly. However, the model trained from scratch (LN & RescaleViT) can not achieve the same high accuracy. It can converges smoothly and achieve a pretty reasonable good result because it is trained on a smaller dataset than ImageNet-21k. Our proposed RescaleViT without normalization can achieve the same accuracy compared with the model using layer normalization and our model converges slightly faster.

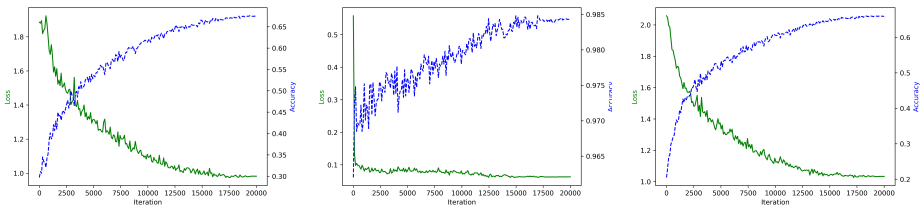


Figure 7: Loss and Accuracy of LN (left), LN*(middle) and RescaleViT(right).

5.4 Discussion

We can observe that the LN* model with pre-trained checkpoints initialization achieve the best result, achieve more than 90% accuracy at the beginning of training and can get more than 98% accuracy. This is because ImageNet-21k dataset is much bigger than CIFAR-10 dataset.Transformer has great

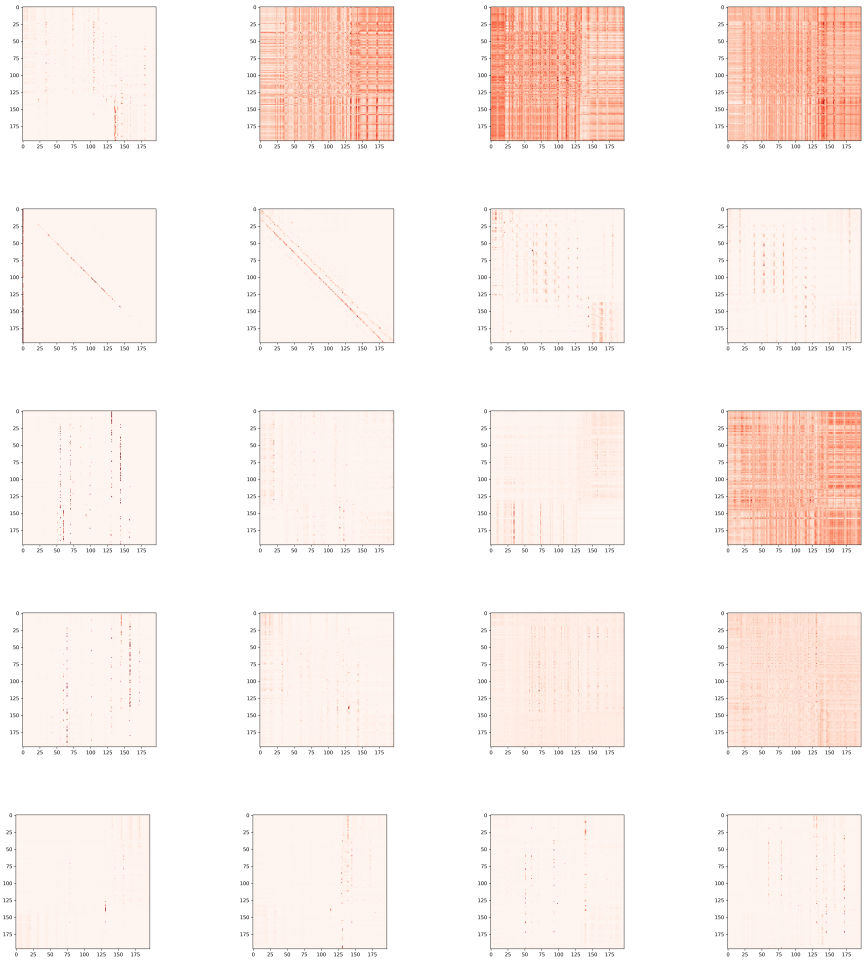


Figure 6: Attention Map. Each row contains the attention map which generated from 5 different model. The first row is generated from a model with layer normalization. The second row shows the attention map from pre-trained model. The model of the third row using batch normalization and the forth one using group normalization. The last row is generated form our RescaleViT which without normalization. And each columns represent different blocks. The first column is the 0 block, the second column is from the 3rd block, the third column is from the 6th block and the final column is from the 9th block.

potential to perform better than CNN when we can provide very large amount of data. Which means there are more parameters and needs more training time.

LN model outperforms than other models almost all the time. Usually we will use batch normalization in computer vision tasks when we use CNN models. However, in our case LN is much better than BN. One possible explanation is BN works badly when the batch size is small. LN works well on both small and big batch size. That's because BN is normalization between samples, small batch samples can't provide a good estimate of the whole dataset. And the statistics within one sample is more important than the statistics within a batch in this case.[12]

Our RescaleViT can achieve the same performance compared with normalization models and our model reduces the computation cost by removing the normalization layers. Because of this, RescaleViT can train faster than other models.

6 Conclusion

In this work, we explored how to train transformer without normalization layer and can achieve the same performance. We discussed the gradient exploding and vanishing problem in train deep neural network. We proposed RescaleViT based on vision transformer. Then we conducted a series of experiments to test our method. The result shows that transformer has potential to outperform than CNN but requires large amount of data. In future work, we will investigate more on why normalization helps training network and why LN is better than other methods in transformer.

References

[1] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. 2016.

[2] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. 2018.

[3] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. 2020.

[4] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pages 580–587. IEEE, 2014.

[5] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. 2015.

[6] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778. IEEE, 2016.

[7] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. 2015.

[8] A. Krizhevsky and G. Hinton. Learning multiple layers of features from tiny images. *Handbook of Systemic Autoimmune Diseases*, 1(4), 2009.

[9] Nvidia. Apex: A pytorch extension: Tools for easy mixed precision and distributed training in pytorch. `https://github.com/nvidia/apex`, 2020.

[10] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 779–788. IEEE, 2016.

[11] Jie Shao, Kai Hu, Changhu Wang, Xiangyang Xue, and Bhiksha Raj. Is normalization indispensable for training deep neural network? In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 13434–13444. Curran Associates, Inc., 2020.

[12] Sheng Shen, Zhewei Yao, Amir Gholami, Michael W Mahoney, and Kurt Keutzer. Powernorm: Rethinking batch normalization in transformers. 2020.

[13] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. 2014.

[14] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. 2014.

[15] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–9. IEEE, 2015.

[16] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. 2017.

[17] Yuxin Wu and Kaiming He. Group normalization. In *Computer Vision – ECCV 2018*, Lecture Notes in Computer Science, pages 3–19, Cham, 2018. Springer International Publishing.

[18] Hongyi Zhang, Yann N Dauphin, and Tengyu Ma. Fixup initialization: Residual learning without normalization. 2019.