DyF : Dynamically Freezing ViT Blocks in Fine-tuning

Sehee Kim [1]

1. Introduction

When we fine-tune ViT-based models, we usually use the original ones whose blocks are all activated to update parameters, or the revised ones some of whose upper blocks are inactivated. Inactive blocks could be efficient because they save computational resources by skipping gradient updates. But both methods are static. Even if you inactivate some layers, you should decide how many layers are frozen.

Having decided whether some layers are frozen or how many layers are frozen, you have to make a decision again. How many epochs would run in fine-tuning? We usually stop fine-tuning when no more validation loss is updated. This method depends on the data used to validate. If there are some problems with the validation data, the fine-tuning could be affected and may have low accuracy.

2. Related Works

2.1. Vision Transformer

Vision Transfomer(ViT) was introduced by Dosovitskiy et al. Originally, transformer is used in language models. The paper adopted transformer to image model for the first time. ViT divides images to patches like tokens in natural language processing(NLP).

Classical CNN-based models must have deep networks to global and local information. Compared to this, ViT doesn't need too deep networks. Instead, it uses self-attention which make possible to learn global interaction of images efficiently. It means that ViT is good at training using a huge dataset. And it shows high efficiency in transfer learning.(Dosovitskiy, 2020)

2.2. Transfer Learning

Transfer learning refers to the process of re-training a pre-trained model on a specific dataset for a new task. This process is called fine-tuning. ViT requires huge dataset at pre-training, but in fine-tuning it doesn't require huge one. Just small dataset is sufficient.

When the user fine-tunes, he or she can use all the blocks or can 'freeze' some blocks. Freezing means that the user decides not to update parameters in certain layers. fine-tuning is mainly needed to update deeper layers which reflect detailed features because general feature is trained when the model is pre-trained.(Houlsby et al., 2019)

2.3. Early Stopping

Early Stopping is the method that if there is no more validation improvement, training is 'early stopped'. Patience parameter decides how many epochs will be maintained without validation improvement. This method can prevent overfitting. But it depends on validation data. If that data can't represent all data, it can distort the timing when early stopping occurs.(Bengio, 2012)

Algorithm 1 *DyF* Algorithm

```
Input: model, threshold
epoch= 0
All the blocks are unfrozen
repeat
  epoch=epoch+1
  for i = 1 to 12 do
    Calculate average gradient
    if average gradient > threshold then
      i^th block = frozen
    end if
  end for
until all the blocks are frozen
```

3. Method

There are basically 12 blocks in transformer in ViT. But when fine-tuning, all the blocks don't need to be updated because general information is pre-trained, and only several detailed features are updated. In this study, I propose *DyF*, a method where each block is dynamically frozen based on a threshold, and training stops when all blocks are frozen. Algorithm 1 directly shows how it works in real code. In this algorithm, the average gradient of each block is the comparison standard to the freezing threshold.

*Equal contribution [1]Department of Computer Science and Engineering. Correspondence to: Sehee Kim <sehee020512@korea.ac.kr>.

Proceedings of the 38^{th} International Conference on Machine Learning, PMLR 139, 2021.

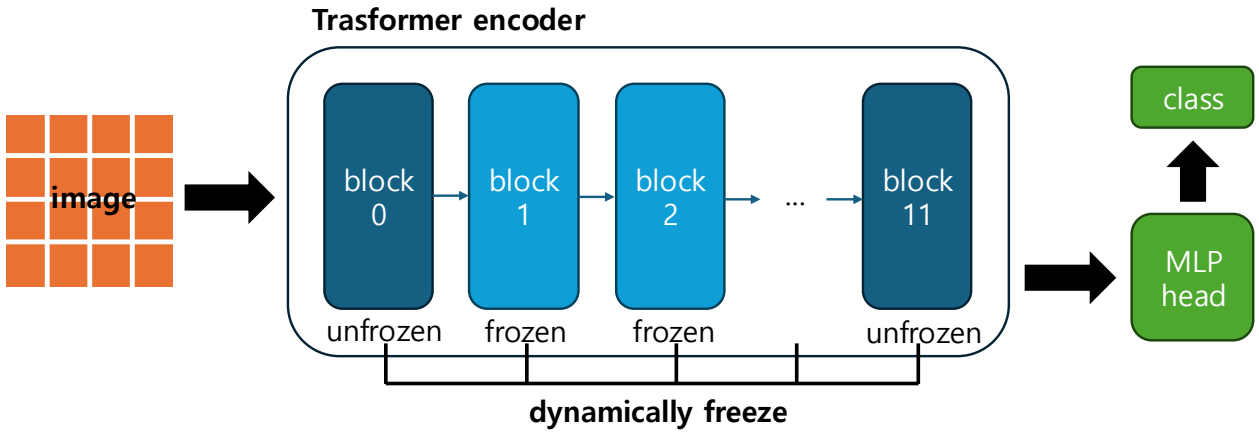

Figure 1. Example pipeline of frozen blocks of ViT in fine-tuning

You can see the pipeline of one example during *DyF* in Figure 1. Some blocks are frozen, and the others are unfrozen. Generally, the upper layer would be frozen first and the deeper layer would be frozen later. But it can differ according to the features of the dataset *DyF* can reflect it. So, in the frozen blocks the parameters are not calculated when back-propagation. They are calculated only when forward-propagation. It could save various resources.

I expect the following advantages from *DyF* 1) It can decrease training time because the process just goes through frozen blocks when back-propagation. It only calculates the parameters of unfrozen blocks. In the same way, 2) it can decrease usage of GPU RAM. 3) It can automatically finish fine-tuning. At the certain point where all the blocks are frozen, it stops.

4. Experiments

4.1. Preliminaries

Dataset I used *Hair Type Dataset*(Sree, 2024) in this experiment. It has about 2K images and 5 classes with labeling which are straight, wavy, curly, dreadlocks, kinky. The images capture a person's hair. So it may be proper to fine-tuning for common usage.

Computer Resource This experiment was run on Colab using A100 GPU.

4.2. Experiment Design

I designed the overall experiment to compare fine-tuning with Early Stopping and fine-tuning with *DyF*.

Data Preparation The dataset is divided as 70% for training, 20% for validation, and 10% for testing and each image is labeled by class. Fine-tuning with early stopping uses the entire dataset, but fine-tuning with *DyF* uses only training and testing dataset except validation.

Table 1. Quantitative Results from fine-tuning with Early Stopping

EPOCH	VALID LOSS	DURATION(S)
1	0.3748	25.74
2	0.4754	24.84
3	0.8788	24.74
4	0.3200	24.85
5	0.3505	24.98
6	0.3564	25.28
7	0.3946	24.99
TEST ACCURACY: 90.55%		175.42

Fine-Tuning with Early Stopping Patience parameter is set as 3, which means if there is no improvement in validation loss for 3 epochs, it stops current training. This method compares the lowest loss and current loss. If the current loss is lower than the minimum, it updates and saves the current model as the best one.

Fine-Tuning with *DyF* The overall process is the same as Algorithm 1. But freezing threshold must be set, which decides whether the block is frozen or not. I set the threshold as 0.001 through some tests. It would be compared to the average gradient of the block in each epoch. *DyF* doesn't need to calculate the validation loss.

4.3. Quantitative Results

Fine-Tuning with Early Stopping Table 1 shows the quantitative result using this method. The training ran with similar durations for every epoch. Because validation loss isn't improved for epoch 5-7, the final model is the one in epoch 4. It produces an accuracy of 90.55% for the test.

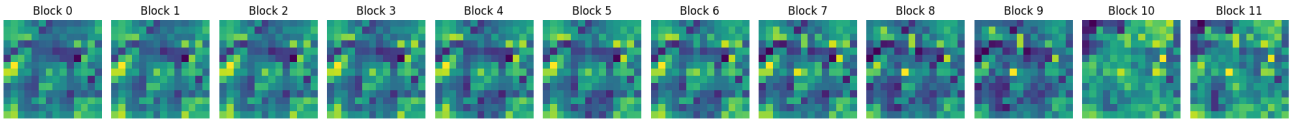

Figure 2. Feature maps of each block after fine-tuning with early stopping

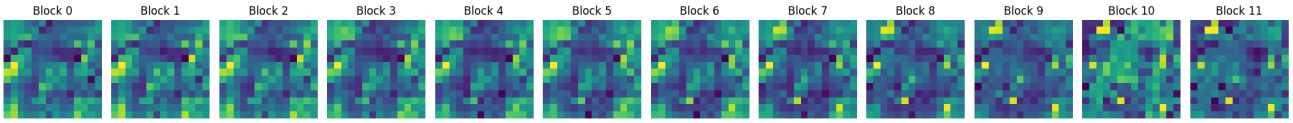

Figure 3. Feature maps of each block after fine-tuning with *DyF*

Table 2. Quantitative Results from fine-tuning with *DyF*

EPOCH	TRAIN LOSS	DURATION(S)
1	1.1031	21.43
2	0.3293	21.24
3	0.1541	21.28
4	0.1165	21.26
5	0.0406	21.19
6	0.0435	21.33
7	0.0179	21.16
BLOCK 6, 8, 9, 11, 2, 3, 4, 5, 7 IS FROZEN		
8	0.0076	20.63
BLOCK 1, 0, 10 IS FROZEN		
9	0.0040	18.55
TEST ACCURACY: 93.53%		188.07

Fine-Tuning with *DyF* The full operation of fine-tuning with *DyF* is seen in Table 2. We can see an explicit difference decrease in duration after freezing. In epochs 1-7, the duration is around 21 seconds. But after freezing, it shows an obvious jump to around 20 and 18 seconds. Corresponding to blocks getting frozen, the time gets shorter in one epoch. Also, the train loss gradually decreases.

Comparison We can see several differences between both methods from the tables. The classical way uses fewer epochs but takes more time in each epoch. Furthermore, the durations are almost the same in every epoch.

The way with *DyF* uses more epochs but takes less time in each epoch gradually decreasing the length of duration. As a result, test accuracy has no big difference but the *DyF* shows a slightly higher accuracy than the classical one. Still, the entire duration is longer in *DyF*.

4.4. Qualitative Results

Figure 3 shows more differences among the feature maps of each block than Figure 2. It means a model with early stopping learns at similar levels over blocks. On the other hand, the model with *DyF* learns various levels of pattern effectively.

5. Conclusion

This study suggests that *DyF* which is the method that dynamically freezes the blocks of ViT in fine-tuning and stops training when all blocks are frozen. It uses less time in each epoch and the duration decreases after every freezing. It gets an accuracy improvement of 2.98% with only a 7% increase in overall duration compared to the classical method.

Also, *DyF* can adapt to various datasets. For the dataset with locally detailed features, it may freeze the upper layers and primarily fine-tune the deeper layer. On the other hand, for datasets with globally diverse features, the model may freeze blocks at later epochs compared to those with detailed datasets.

From these, *DyF* can be one of the successful methods.

6. Future Direction

Various Datasets In this study, only one dataset is used. Various datasets with diverse sizes, features, etc. are needed to validate whether *DyF* is practically effective. Huge datasets might show obvious decreases in overall durations and GPU RAM, which aren't seen in this study.

Various Methods Only early stopping is handled in this study to compare. There is another method in transfer learning that statically freezing blocks. It freezes several upper layers before starting fine-tuning. Comparing various methods gives a more accurate characteristic to *DyF*.

Freezing Threshold I selected the freezing threshold as 0.001 through some tests with different thresholds. Automatically deciding the threshold might be helpful for the user's convenience. Finding how the threshold should be selected is needed.

References

Bengio, Y. Practical recommendations for gradient-based training of deep architectures. In *Neural networks: Tricks of the trade: Second edition*, pp. 437–478. Springer, 2012.

Dosovitskiy, A. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

Houlsby, N., Giurgiu, A., Jastrzebski, S., Morrone, B., De Laroussilhe, Q., Gesmundo, A., Attariyan, M., and Gelly, S. Parameter-efficient transfer learning for nlp. In *International conference on machine learning*, pp. 2790–2799. PMLR, 2019.

Sree. Hair type dataset. https://www.kaggle.com/datasets/kavyasreeb/hair-type-dataset, 2024. Accessed: 2024-10-16.