SUPERPIXEL-BASED EFFICIENT TRANSFORMERS

Yu Qi
Peking University
Beijing, China
1900012985@pku.edu.cn

It summarizes the thoughts and works, including future tryouts of mine in the summer of 2022. The computational resources is not enough (4*TITAN xp) to run dense experiments of transformers on large classification datasets or use large models, moreover, due to some GPU issues and changes of high-level directions, currently the project is **not finished yet** and experiments are very few and toy. However, I am very happy to write this draft because it records some thoughts that I think is very worth formulating, sharing and discussing. Also, it is a great milestone of mine to think and conduct research situated in the intersection of statistics and computer science more independently.

Hope this draft could be beneficial for future studies of mine or others.

Please see more in Acknowledgement. 4

ABSTRACT

Superpixel is a tool that merges different small pixels with similar locations, textures, and colors into bigger ones. Transformer is a strong model that is introduced to vision from the field of Natural Language Processing. After some unsuccessful tryouts in the stem layer, the design and verification of a theoretically-proved very efficient attention mechanism under the high-level guidance of superpixel is currently on-going. In the meantime, other thoughts and projects about superpixel are brought up.

***K*eywords** superpixel · transformer · efficiency · segmentation

1 Introduction

1.1 Superpixel and its generation methodologies

Superpixels, as a tool to aggregate pixels that share silimlar locations, texture and color, is a concept that is firstly brought up in the year of 2003 when deep learning hasn't overwhelmed the classical algorithms of computer vision [1]. It focuses more on rigion-level rather than pixel-level of an image, uses a representitive RGB information to represent all other pixel-level information. it is a great way to reduce the complexity and redundent information of original images, therefore often serves as a popular data-preprocessing way for various downstreaming tasks.

Methodologies for generating superpixels are various. There are both non-supervised ways like clustering methodologies, and learning-based methodologies. The most popular one that generates supeprixel off-line is SLIC [2], it uses revised $K-means$ algorithm and introduces the idea of local-window to make sure the mathematical complexity of the whole algorithm is $\mathcal{O}(n)$, and n represents the number of superpixels. There are also other ways to generate superpixels, like graph-based methodologies (***Lemma1*:** The non-rigid format of superpixels naturally aligns with topological graph, each node represents a superpixel, and the edges represent the neighboring relationships between those superpixels).

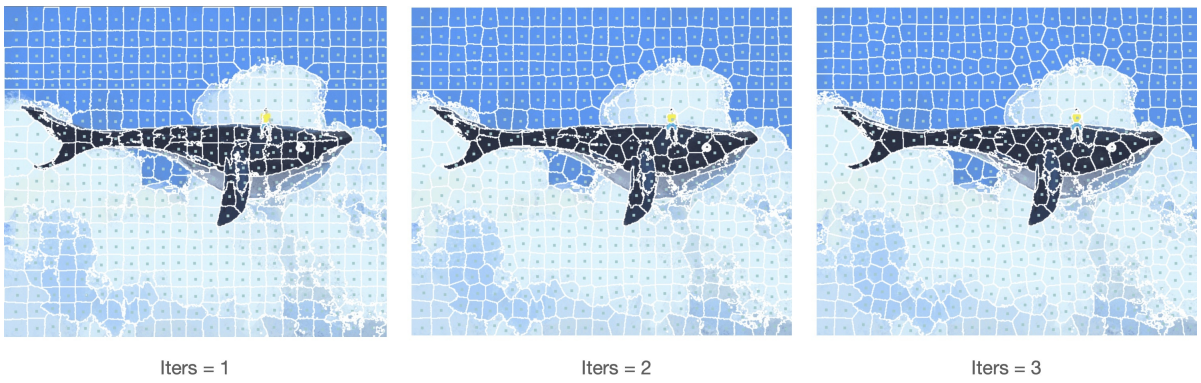

Figure 1: SLIC - superpixels
As shown in the graph, each region representing a superpixel. And each small node in the region represents the cluster center of that superpixel, whose RGB value will be representitive of all pixels in that region.

1.2 Transformers and its common variants with positional embedding

Transformer [3] occurs with the concept of 'attention-mechanisms', in which each datapoint x in the space X can find relationships with each other after linear projection into another space like Q, K, V(The projection is done by the multiplication factor of W_i,W_j,W_k). The design of three different linear projections is for the introduction of additional learnable parameters to increase the generalization of learning.

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

Cross-attention and Linear-attention Apart from the above self-attention mechanism, there are also other kinds of attentions like cross-attention, where the data-points are from different spaces rather than the same one. Apart from attention in spatial-dimensions (e.g. the $H * W$ in $(H * W * C)$), there are also kinds of attentions in channel-dimensions, the most popular method among which is the squeeze and excitation network [4]. Meanwhile, to reduce the $\mathcal{O}(n^2)$ complexity that is brought by the matrix multiplication, there are also ways [5] about linear complexity, the mathematical principles are listed as follows, all they are trying to do is to try to find a 'linear attention' that can reach the similar performance with 'vanilla attention', however that suffers from bottlenecks because it brings lower rank of matrix, which represents less effective ability to deal with information, and there is a paper further illustrated about this issue. ***Lemma2*:** So linear attention is still in the area of Natural Language Processing and not widely applied in the field of computer vision because it doesn't receive very good effect.

$$Attention(Q, K, V)_i = \frac{\sum_{j=1}^{n} sim(q_i, k_j)v_j}{\sum_{j=1}^{n} sim(q_i, k_j}$$

The positional embedding is a trick that firstly occurs in the field of Natural Language Processing. It is very clear that in the vanilla attention the position of each element is not influenctial to the final result (for example, if we randomly shuffle those datapoints, the results can still be very much alike). But in natural languages we know that the position of each word vector really matters (e.g. Bob is the father of David *v.s.* David is the father of Bob). So the injection of positional embedding, or $P.E.$, is desiged to resolve this kind of problem. In computer vision, many people copy that kind of design aim to enhance the spatial relationships between each patch of the image, but it receives improvement not that obviously. ***Lemma3:*** Just like 1.1 ***Lemma1*** said, the superpixel is non-rigid, which breaks up the regularity in the $Euclidean\ Space$, so hopefully the injection of positional embedding may reduce that kind of damage and improve the performance. The work of swin transformer [6] is an example to inject positional embedding in computer vision, where B represents for the relative position encoding:

$$Attention(Q, K, V) = SoftMax(\frac{QK^T}{\sqrt{d}} + B)$$

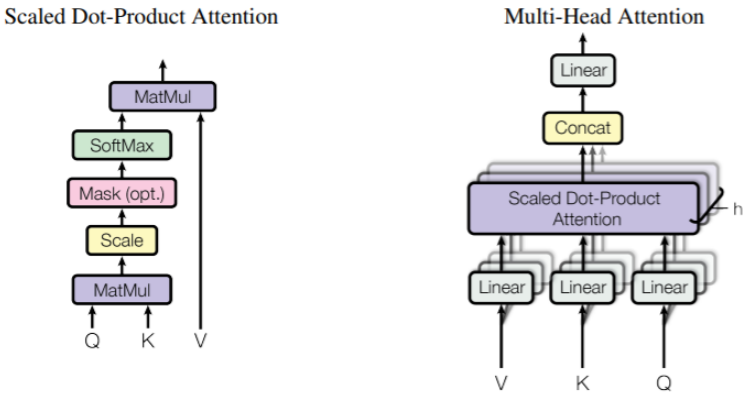

Figure 2: attention mechanism

1.3 The correlations between self-attention and superpixels

$$x_{ij} = \alpha_i \cdot \beta_j \in Q * K^T$$

It can be clearly seen that $\alpha_i \cdot \beta_j$ measures the similarity between two vectors: α_i and β_j, it is not a number between $|\alpha_i| * |\beta_j|$ and $-|\alpha_i| * |\beta_j|$, can be interpreted as a variant of cos function. After all vectors x are through the $Layer\ Normalization$ and get the similar length, the dot multiplication can measure the similarity very precisely.

As algorithms in SLIC (we will leave the details of the algorithms), the pixels are all transformed from the space of RGB to the space of $CIELAB$. And there is a distance function $dis(x, y)$ to assign different weights to parameters in $CIELAB$ color space, and the x, y position of each pixel. So it is very natural to think that

$$dis(x, y) = C * \frac{1}{x \cdot y}$$

where C is a constant, and $\cdot$ can be measured by self-attention mechanism.

Different from the off-line unsupervised clustering methodologies like SLIC, self-attention enjoys great superiority because it provides learnable weights to make update possible.

Proposal1: But that is a little off the initial proposal now, because in this way we bring up a new method to achieve segmentation like superpixel, rather than to bring efficiency in transformer.

1.4 The motivation of superpixel-based efficient transformers

As described in 1.1, superpixel is a way to remove redundancy in an image. The motivation is very clear: we can only use transformers to handle those extracted datapoints from the original image space, put limited computational resources on the smaller number of data that is representative. We can reduce the number of data in spatial-dimension, and enhance the number of C in channel-dimension, to see if the performance is similar as or better than previous methodologies. In this way we can build a superpixel-based efficient transformers that the industry will love.

2 Methodologies

To achieve the ambitious goal of efficiency, there are many possible ways can do this: some are on stem layer of a neural network, while the others are on the block.

2.1 Block

It is very natural to design a thin 'bottleneck' insert it in the neural network to aggressively merge features into 'superfeatures', and in the latter part of neural network, because of the reduction of the number of features in spacial dimension, we can increase the channel dimension to achieve similar performance with lower cost of computational resources.

2.1.1 A potential and better way to merge tokens

Because there are already many existing works to merge tokens, but the performance of them are not that good. Almost all of them use attention-variants to do this. So maybe a way that merges tokens using superpixel-based methods can be designed as a plug-and-play module in the future, and thanks to my advisors for bringing it up to me. But that may require further design, because I personally think there is very limited space to improve the methodologies in 'learning to merge tokens' [7], hopefully the future readers could give me some advice for this.

2.1.2 A mathematically-efficient transformers

Motivation

Superpixels are to some degree alike the Hierarchy of the Computer System. **Layer1:** The normal and common pixels. **Layer2: or (higher-layer)** The representative pixels, also named the cluster center. It seems that cluster centers is more abstract, and enjoys more representative meaning, and inspired by this we can use similar **hierarchy** ideas as the principles in computer architecture, let the representative pixel to update each pixel inside the local region, rather than let all pixels in one local region to do self-attention.

It seems that because the shape and size of superpixels are **irregular**, and the generation of superpixels, no matter supervised way or un-supervised way, is **time-consuming**, even if it is SLIC, as we have already tested it in 2.2.2. So we consider the superpixel (or other clustering methodologies) is more a way to achieve higher performance, rather than efficiency.

Mathematical formulations and previous methods

Every variant of vanilla attention, as long as it updates all original features from the space of X, the FLOPs (Floating Points Operations) are strictly limited by this formulation.

NOTICE: *Please recall the definition of Ω, the exact value maybe not very accurate, for example there are operations on channels like multi-head that could influence the overall computational complexity. Complexity comparisons between different attention and the special case to achieve the equation is absolutely precise.*

$$\Omega(attention) \geq 2hwC + (2 + hw)C^2 \tag{1}$$

The equation is achieved when K is the projection of original feature space X, and Q, V is the projection of one vector $(1 * C)$. This situation means that we only use one tensor to update the original feature space.

A vanilla attention suffers from $\mathcal{O}(n^2)$ computational complexity:

$$\mathcal{O}(vanilla\ attention) = 2h^2w^2C + 3hwC^2 \tag{2}$$

The swin [6] attention is still $\mathcal{O}(n^2)$, but the introduction of local window brings linear reduction, and M^2 means the number of local windows:

$$\mathcal{O}(swin\ attention) = 2M^2hwC + 3hwC^2 \approx 98hwC + 3hwC^2 \tag{3}$$

The attention in Pyramid Vision Transformer [8] firstly uses convolution to do spatial reduction when generating K, V. But the overall attention is still in $\mathcal{O}(n^2)$. The newest version of PVT, called PVT-v2 [9], firstly use pooling as a downsampling methodology to achieve linear attention in computer vision:

$$\mathcal{O}(pvt - LSRA) = 2P^2hwC + (hw + 2P^2)C^2 \tag{4}$$

where P^2 denotes the number of remaining features after pooling, rather than the size of pooling kernel, in this case $P = 7$, so $\mathcal{O}(LSRA) = 98hwC + (hw + 98)C^2$

Our methodologies

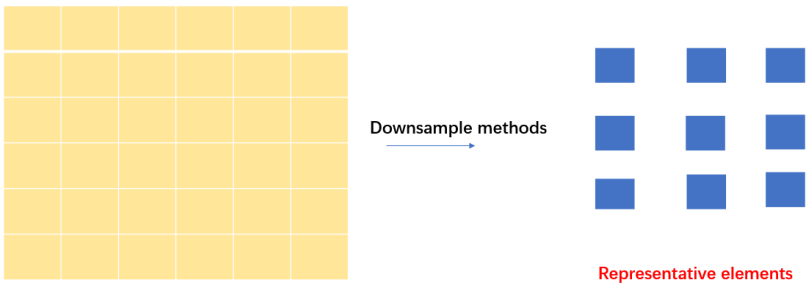

Figure 3: downsampling methodologies
In this picture, we use average pooling as the simplest way for downsampling to see what kind of efficiency we can get at most.

Inspired by the hierarchy system, we build a series of attentions that enjoys linear complexity. To make it more simple, we remove the irregularity brought by superpixels, choosing adaptive average pooling as the simplest way for clustering to achieve the most efficiency. (Pooling can be interpreted as a simplest way of clutering).

In the future, I plan to use another differentiable downsampling methodologies to compromise efficiency and boost performance. So this 'pooling strategy' aims to be a toy example to test how efficiency we can get at most. See Fig. 3, and the mathematical proof about why this kind of strategy is too trivial so it is ignored.

Terminology

original feature $N * C = HW * C$
cluster center $M * C$
downsample method $AdvPooling()$
local region $R * R$
number of partitioned windows $P * P$

A general case to perform this hierarchy linear attention is listed as follows:

Algorithm 1 general case of a hierarchy linear attention

INPUT $N * C$
$AdvPooling(N * C) \rightarrow M * C$
$self - attn(M * C) \rightarrow M * C$
$Window - Partition(M * C) \rightarrow P^2 * cluster\_windows$
$Window - Partition(N * C) \rightarrow P^2 * feature\_windows$
(potential) $P.E.$ on $windows$
$cross - attn(P^2 * feature\_windows, P^2 * cluster\_windows) \rightarrow P^2 * feature\_windows$
$reverse - partition(P^2 * feature\_windows) \rightarrow N * C$
OUTPUT $N * C$

And its computational complexity is:

$$2(\frac{NM}{p^2} + M^2)C + NC^2 + 5MC^2 = 2(\frac{M * hw}{p^2} + M^2)C + (hw + 5M)C^2 \tag{5}$$

And there are two corner cases, because I think future modifications will be more on **case1** so I put more words on it.

Case1:

$$\mathbf{2} * (hw + M^2)C + (3M + hw + 2P^2)C^2 \tag{6}$$

Case1 is when the cross-attention is happening between each local region of original features and only one corresponding cluster center. Clearly this one enjoys really high-efficiency.Because most of my future modifications

For example, if we set the pooling kernel size to be $(8, 4, 2, 1)$, $P = (7, 7, 7, 7)$ just as the PVT-v2, then the value would be :

$$\mathbf{2} * (hw + 49^2)C + (5 * 49 + hw)C^2 \tag{7}$$

And it is a really small number compared with pvt-v2, because we've already made big process by reducing the complexity from $\mathcal{O}(n^2)$ and $\mathcal{O}(98n)$ to $\mathcal{O}(2n)$. Recall pvt-v2:

$$\mathbf{98} * hwC + (hw + 98)C^2 \tag{8}$$

Especially when h, w is of high resolution like $1000 * 1000$ rather than $224 * 224$, this method enjoys greater supriority than others. And if we recall the theoretical lower bound:

$$\Omega(attention) \geq \mathbf{2} * hwC + (2 + hw)C^2 \tag{9}$$

We will find it is definitely pushing the limit to the theoretical lower bound of attention-mechanism, without using any tricks like astrous convolution to ignore any data.

Case2:

$$2(M * hw + M^2)C + (hw + 5M)C^2 \tag{10}$$

Case2 is when the cross-attention is happening between whole original features and whole cluster centers. Clearly this one have worst theoretical FLOPs, but maybe greater performance.

Meanwhile, there are also other methods between those two cases to make compromising between efficiency and performance. I will leave more details in the supplementary materials.

Dataset and task

We use imagenet-100 [10], a subset sampled from imagenet-1k [11], and train the model for classification tasks for 200 epochs.

Experiment Result

Table 1: Experiment Result on Imagenet-100 Classification

Method	Pooling Size	Size1	Size2	Param	GFLOPs	Acc1-100	Acc5	GPU Latency	CPU
MobileViTv2-1.0	N/A	-	-	4.9	1.8	**89.63**	98.11	7.67	29.13
EfficientNet-b1	N/A	-	-	7.8	0.7	87	**97.64**	14.78	28.4
PVTv2-b0	N/A	-	-	**3.4**	0.6	82.8	95.86	**5.79**	**19.95**
PVTv2-b0-li	[7,7,7,7]	-	-	3.39	0.56	81.42	95.6	6.63	21.77
Our-b0	[7,5,4,3]	-	-	3.98	0.53	80.86	95.38	7.64	22.48
Our-b0-1	[7,5,4,7]	-	-	3.98	0.57	81.38	95.6	7.70	23.36
Our-b0-2	[7,5,7,7]	-	-	3.98	0.59	81.88	95.42	7.67	21.93
Our-b0-3	[7,7,7,7]	-	-	3.98	0.59	81.86	95.58	7.46	23.10
Our1-b0-1	[7,7,7,7]	[8,4,2,1]	2*2	3.98	0.61	81.96	-	26.59	-
Our1-b0-2	[7,7,7,7]	[8,4,2,1]	**1*1**	3.98	0.55	81.70	-	8.62	24.29
Our1-b0-2 w/o self-attn	[7,7,7,7]	[8,4,2,1]	**1*1**	**3.4**	**0.52**	79.62	-	7.44	21.98

Pooling size: P, P^2 represents the number of representatives at four different stages.

Size1: S_1, S_1^2 represents the size of each local window on original features when doing cross-attention.

Size2: S_2, S_2^2 represents the size of each window on cluster centers when doing cross-attention.

GPU Latency(ms): is tested on 4*titan xp with batch_size = 1, iterations = 300

CPU Latency(ms): is tested on the CPU of CCVL10

Due to some GPU issues, current meaningful experimental results are very few.

Analysis and ablation studies

1.From the result of Acc1-100 on 'Our1-b0-2' and 'Our1-b0-2 w/o self-attn' we can see that the first layer: 'self-attn' layer brings the global receptive field, and is important to the overall performance. It proves the effectiveness of this hierarchy systems using double attention.

2.From the result of Acc1-100 on 'Our-b0', 'Our-b0-1', 'Our-b0-2' and 'Our-b0-3', we can see that the last stage, or stage4, has more meaningful semantic information than stage3 and stage4.

3.From the result of GFLOPs of 'PVTv2-b0-li' and 'Our1-b0-2', whose attention cost is theoretically calculated and compared in formulation (7) and (8). We can see at the setting of PVTv2, with input image resolution equals to $224 * 224$, the reduction of FLOPs is not obvious, which is very curious. **There are following reasons that could cause this phenomenon:**

a) The input resolution $hw = 224 * 224$ is too small, maybe enhance the resolution could achieve obvious improvement on FLOPs

b) The reduction of the FLOPs on attention doesn't affect much on the overall FLOPs of neural networks, considering there are other parts like MLP which is FLOPs consuming.

c) Maybe we should test FLOPs on other scales of network rather than B0 to see if there are better FLOPs reduction.

Future Plans

Test what causes the huge gap between the theoretical calculation and the actual performance of FLOPs to achieve efficiency.

Then, boost performance maybe by changing the downsampling strategy from pooling to 'deformable convolution' or other strategy to downsample elements like the feature version of superpixel. And add some additional trisks like positional embedding to learn more positional information.

2.2 Stem layer (honestly speaking, I don't like those of my modifications, please move to the other part to find more interesting things)

As the first barrier between the input and the output, stem layer is fed directly by the input image. So it is very natural to firstly reduce the redundancy of an image in the stem layer before the block. Usual stem layers are designed as a vanilla convolution with a 3*3 or other size of kernel to downsample an image. A few refined designs of mine are separately added to it to enhance the filtering function of the stem layer.

2.2.1 Cluster-center amplifier

Motivation

As in SLIC [2] algorithm, each region representing superpixel has a central datapoint, we call it 'cluster center', which represents the entire information in that region (or cluster). It is natural that we can use those cluster centers to represent all other pixels, to downsample an image in the other way.

Methodologies

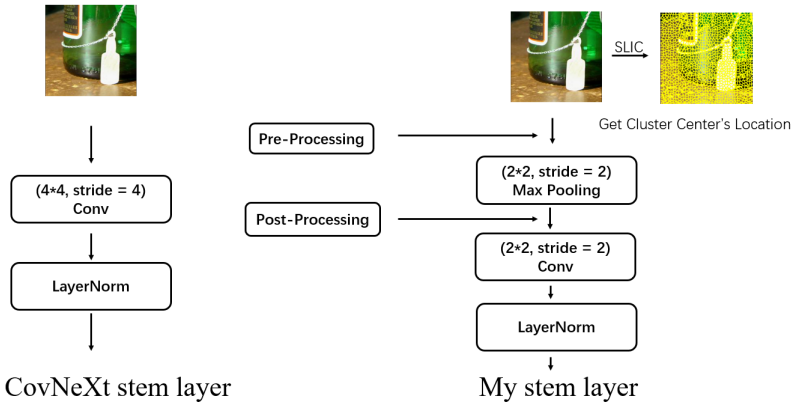

Figure 4: modified stem layer1

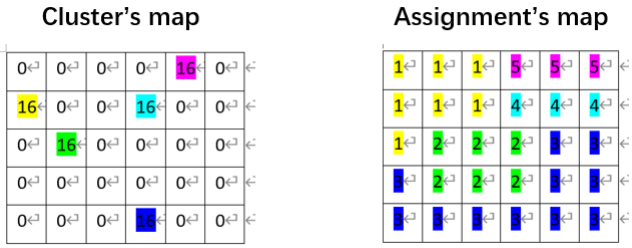

Figure 5: design of cluster-map
After using SLIC to pre-process the whole dataset, we get the locations of cluster center and build a cluster-center-map, where the positions of each cluster center is 16, the other positions are all 0.

Proposal2: Please see the rough methodology in Fig 4. Apart from word illustrations as follows, a pseudo code is also provided.

a) SLIC

SLIC here is a data-preprocessing methodology, can be interpreted as a kind of data-augmentation, such as other transforms in the library troch.vision. We think that the RGB information of each cluster center is representitive and useful, and see information of other pixels as redundant, so what we are trying to do here is to locate the position of each cluster center and design the map of cluster center, see in Fig 5

b) Pre-processing

After getting the location of each cluster center, what we want to do is to select them and remove the rest of other pixels, in the meantime and more importantly, to save the euclidean regularity of each image, rather than turning pixels into a topological graph and damage euclidean regularity of an image.

'Max pooling' here serves as a great filter to move away redundant pixels and select the most representative ones.

However, because the locations of cluster centers are scattered and suffers from the irregular distributions of positions, there must be a compromise between the saving of more cluster centers and the regular distribution of the saved pixels.

So what we are trying to do here is to build an 'amplifier' based on Max Pooling, to give a more superiority to cluster centers, and make them more likely to be saved when doing pooling. Therefore we make sure the patches are rectangular-distributed, in the mean time save more meaningful information.

Please see more details in Fig 6

c) Max Pooling

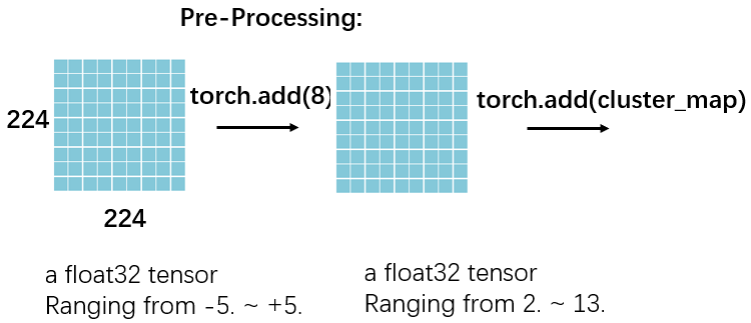

Figure 6: pre-processing
Firstly we add 8 to all the data to get all of them as positive numbers, which is convenient to do mod functions. Then we add cluster map to give special locations a high-prior before downsampling them. Those numbers are designed by analyzing the numerical distributions of the data in Imagenet-1K [11], meanwhile using the power of 2 to lower down the precision damage on original data based on the format of floating point in IEEE-754.

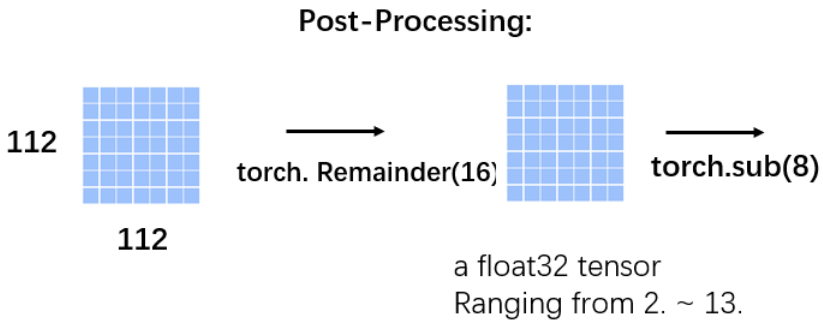

Figure 7: post-processing
As is similar in Fig 6, after pooling, we need to recover those modified data as their original value to make sure our marks have no influence on those features.

Max Pooling here serves as a square filter to downsample an image, we use 2*2 kernel size so here we only get $\frac{1}{4}$ useful information, it is a very small number compared with the number of all pixels in an original image. We would like to see that if those selected and more representative pixels could replace the information of whole image.

d) Post-processing

After maxpooling, we need to make sure our 'prior' on those pixels are not influential on the following methods, so it is necessary to recover them, as shown in Fig 7.

e) Convolution

After those modifications, we can do 2*2 normal convolution to pachify those selected pixels.

Some frequent questions and answers

Q1: How would you promise that the cluster center can be selected when doing 'max pooling'?

There's no guarantee that cluster center must be selected when doing 'max pooling', we just designed a tool to give cluster center a kind of prior, to make them more likely to be chosen when doing 'max pooling'.

To be more specific, there are four kinds of situations when doing max pooling, please see figure 8 for detailed illustrations.

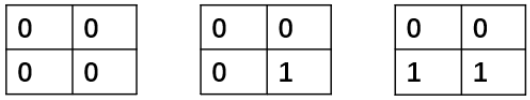

Figure 8: three different cases when pooling

Zero denotes for non-cluster-center pixel, and one denotes for cluster-center. Three cases separately represent zero, one and more than one cluster center inside a single kernel when doing max pooling. Clearly in the left case, it's just non-cluster-center pixels so it will select the maximum of them. As for the middle case, the cluster center would be selected, which is what we want. As for the right case, two cluster centers has superiority, so it will be the maximum between those two cluster centers. Therefore, our method only give cluster center a kind of 'superiority', make them superior when competing with non-cluster-center pixels, but that doesn't necessarily mean that they will be selected. And this design is to save the regularity of Euclidean Space in an image.

Experiment Result

Please see table 2.

The missing epochs and future improvements

Initially I used the dataset of ImageNet-1k (didn't use the subset of it), making the training very slow on current computational resources. Meanwhile the python version of SLIC is a clustering method, and it is super **slow** though it is a $O(n)$ algorithm, so I only get very pitiful experiment result on it, and it's not promising so I cut this experiment off.

I made improvement by changing the dataset of ImageNet-1K to ImageNet-100. It occurs to me that I should use CUDA-version of SLIC and insert it in my network, but I noticed the bad experiment result before I decided to implement it, meanwhile that requires root authorization that I lacked at that time because of some problems with cmake. And I notice that **those modifications suffer from very very bad research taste**, and discover something more meaningful to do, so I cut off this branch after discussing with my advisors.

Details and ablation analysis

Ours1:

With stem layer of MaxPooling(kernelsize=4*4, stride=4), convolution(kernelsize=1*1)

Ours2:

MaxPooling(kernelsize=2*2, stride=2), convolution(2*2, stride=2)

Ours2 w/o slic:

Remain the tensor modifications, but the cluster center map is replaced by a tensor whose elements are all 16. There is not any cluster center information, to test the precision reduction that is caused by our tensor operations.

Baseline(ConvNeXt):

Convolution(kernelsize=4*4, stride=4)

Analysis:

we can see from Ours2 w/o slic that the influence of our operations on the floating point precision is very very destructive. And I conjecture that the advantages by cluster center information will be tiny compared with this.

Table 2: Experiment Result about Amplifiers on ImageNet-1K classification

The training was so slow (5-6 hours for an epoch), and it affect the running and I shut it down in time. For the reason why it is so slow, see 'conclusion'

Methods	**Epochs**	**Acc1**	**Acc5**
Ours1	0	0.588%	2.376%
Ours2	0	0.624%	2.492%
Ours2(w/o slic)	0	0.652%	2.466%
Ours2	60	<< ConvNeXt	<< ConvNext
ConvNeXt	0	0.63%	2.408%

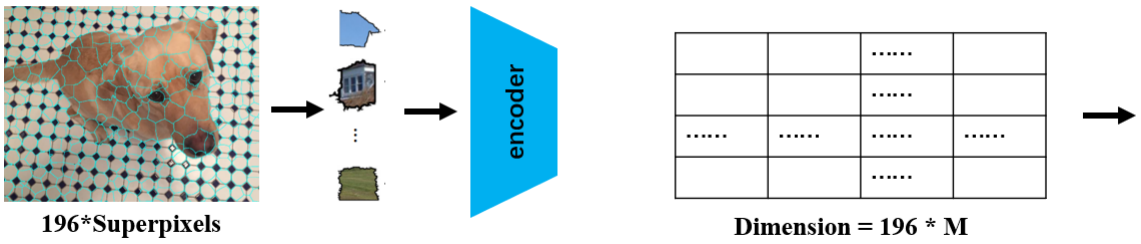

Figure 9: superpixel patch embedding

Conclusion

We find that the result is not very promising, it is very likely because of operations like 'tensor-adding' reduces the accuracy of original data, meanwhile the down-sampling process is too aggressive. The cost of my designed amplifier is too expensive, so in my next modification, I try to save as more data as possible.

2.2.2 Superpixel patch embedding Fig 9

Motivation

Original patch embedding cut images into different rectangular patches very aggressively. There are ways to improve it, for example, deformable patch embedding [12] introduces a new way to learn deformable patch that is more useful to downstreaming tasks. Superpixel is a popular way to segment image into regions with different sizes. ***proposal3:*** So it is very natural to build a new patch-embedding way based on superpixel.

Methodologies

a) data pre-processing

To verify the thoughts, SLIC is used as an off-line methods to generate superpixels.

b) MLP

Because the size of each superpixel is very various, for example, they can be

$$S \in \mathbf{R}^{M*N}$$

where M, denotes as the pixel number in each superpixel region, is various, but the feature dimension N is fixed of each superpixel.

So it is very natural to use a neural network, for example, a MLP to project them into another linear space, where each superpixel can be denoted as a $1 * C$ vector. So the MLP is not only expected to learn the feature of each region, but also the shape, the size of each superpixel.

Collisions

However, when implementing this idea, we find that there is very similar idea in June 2022 that segment images into different regions, and it is also a differentiable version, please see [13] for further details. In the meantime, this idea 2.2.2 is more to improve the performance, rather than to achieve efficiency. **And current stem layer is already very efficient and tiny, although some parameters can be saved, it doesn't make very much difference to the whole network if we don't downsample the image aggressively like 2.2.1 So we change the direction again.**

3 Conclusion and potential projects

We design a kind of attention that can be theoretically-proved very efficient, however receives not very good performance on the testing of FLOPs. The reason for this is still a thing to further discover.

There are also other kinds of projects that are related with superpixels:

a) Like 1.3, we could come up with a way to use superpixels as a better 'plug-and-play module' to merge features than 'learning to merge tokens'.

b) $pixel- > superpixel- > part- > image$ could be a line to tie different kind of supervision on it

4 Acknowledgments

I would like to firstly thank ***Chenglin Yang*** and ***Xiaoding Yuan*** for both their help in my projects and their guidance and implications for my life choices. Actually they should be authors of this, however I drafted it out to clarify my thoughts without asking for their permissions, so I leave out their personal information here.

In the meantime, I want to thank to ***prof. Alan Yuille*** for guidance on my projects and my internship, ***Dr. Zongwei Zhou*** for discussing with me twice, ***Jieru Mei*** for warm-hearted encouragement.

Also, I would like to thank the team in hfai research(which is a self-less foundation providing cluster for people to freely use GPUs and do research) to accept my inquiry to train models on their server. **However, the dataset imagenet-1K on that public server requires special format, and there is some issues about loss back-propagation when training**. I have tried to use a cloud machine, and my own ubuntu system to upload and download dataset for **a large number of days** to solve the problems, but the workers there are too busy to help me with it, and it's a problem that is not easy to handle all these by myself, so unfortunately I haven't successfully enjoyed the convenience that this opportunity brings to me.

References

[1] Xiaofeng Ren and Jitendra Malik. Learning a classification model for segmentation. In *Computer Vision, IEEE International Conference on*, volume 2, pages 10–10. IEEE Computer Society, 2003.

[2] Radhakrishna Achanta, Appu Shaji, Kevin Smith, Aurelien Lucchi, Pascal Fua, and Sabine Süsstrunk. Slic superpixels compared to state-of-the-art superpixel methods. *IEEE transactions on pattern analysis and machine intelligence*, 34(11):2274–2282, 2012.

[3] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[4] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7132–7141, 2018.

[5] Yunyang Xiong, Zhanpeng Zeng, Rudrasis Chakraborty, Mingxing Tan, Glenn Fung, Yin Li, and Vikas Singh. Nyströmformer: A nyström-based algorithm for approximating self-attention. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 14138–14148, 2021.

[6] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 10012–10022, 2021.

[7] Cedric Renggli, André Susano Pinto, Neil Houlsby, Basil Mustafa, Joan Puigcerver, and Carlos Riquelme. Learning to merge tokens in vision transformers. *arXiv preprint arXiv:2202.12015*, 2022.

[8] Wenhai Wang, Enze Xie, Xiang Li, Deng-Ping Fan, Kaitao Song, Ding Liang, Tong Lu, Ping Luo, and Ling Shao. Pyramid vision transformer: A versatile backbone for dense prediction without convolutions. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 568–578, 2021.

[9] Wenhai Wang, Enze Xie, Xiang Li, Deng-Ping Fan, Kaitao Song, Ding Liang, Tong Lu, Ping Luo, and Ling Shao. Pvt v2: Improved baselines with pyramid vision transformer. *Computational Visual Media*, 8(3):415–424, 2022.

[10] Yonglong Tian, Dilip Krishnan, and Phillip Isola. Contrastive multiview coding. In *European conference on computer vision*, pages 776–794. Springer, 2020.

[11] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[12] Zhuofan Xia, Xuran Pan, Shiji Song, Li Erran Li, and Gao Huang. Vision transformer with deformable attention. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4794–4803, 2022.

[13] Yifan Zhang, Bo Pang, and Cewu Lu. Semantic segmentation by early region proxy. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1258–1268, 2022.