CROSS-AXIS TRANSFORMER WITH EMBEDDING IMPRINTS *

Lily Erickson, Emme
Independent Researchers
Minneapolis
lilyerickson.ai@gmail.com

ABSTRACT

Despite lagging behind their modal cousins in many respects, Vision Transformers have provided an interesting opportunity to bridge the gap between sequence modeling, and image modeling. Up until now, however, vision transformers have largely been held back, due to both computational inefficiency, and lack of proper handling of spatial dimensions. In this paper, we introduce the Cross-Axis Transformer. CAT is a model inspired both by Axial Transformers, and Microsoft's recent Retentive Network, that drastically reduces the required number of floating point operations required to process an image, while simultaneously converging faster and more accurately than the Vision Transformers it replaces.

***K*eywords** Computer Vision · Transformer · Retentive Network · AI

1 Introduction

There have been many attempts at processing visual data using neural networks. Convolution Neural Networks use learned kernels to allow local patches of data to share information with each other. ResNets perform a similar role, but along a different axis, attempting to carry a residual version of their state forward.

Recently, transformers have broken onto the scene, which instead of caring strictly about their neighboring features, instead attempt to create a probability distribution over the entire feature space, using a process known as attention.

$$Attention(Q, K, V) = softmax(QK^T\sqrt{dk})V \tag{1}$$

There are a few problems with attention, notably that the softmax operation is computationally expensive, and that expanding the sequence length scales quadratically with its input size. Despite these flaws, however, transformers have shown excellent performance.

The Vision Transformer (ViT) is a model that takes an image, breaks it into patches, and applies the standard transformer attention over the patches to transform that image into an objective function. Classic implementations have typically used the standard sinusoidal positional embeddings, and the typical quadratic attention.

Earlier this year, researchers at Microsoft unveiled their Retentive Network [1], which showcased a version of a language model that ignores Softmax completely, and manages to reduce the computational complexity in their recurrent version by splitting the matrix multiplication steps into separate channels.

We combine these ideas, along with concepts proposed in Axial Transformers [2], to reduce the computational complexity of the large input size of images, while simultaneously showcasing the lack of dependence on Softmax as a whole.

Table 1: ImageNet 1k accuracy after 10 training epochs

Method	Train Acc	Val Acc	FloPs
Classic ViT	0.18	0.14	47.8b
DINOv2	0.257	0.0	40.8b
CAT w/ Imprint	0.641	0.389	31.8b

2 Architecture: Cross-Axis Transformer

We make several changes to the contemporary Vision Transformer. We note that for the purposes of our testing, our model achieved almost 2.8 times the validation accuracy as the baseline vision transformer, while taking half the training time, and requiring only 2/3rds the number of floating point operations.

We'd also like to note that our research is being conducted on a single tower/laptop combo, and so although our results are extremely promising, there is a clear need for much wider scale testing, using compute platforms that are far more powerful than our own, hyper-parameters that are optimized for longer training sessions, and larger training datasets. We publish these findings in the interest of broadcasting the need for such larger scale testing, and in doing so, hopefully define a new paradigm for vision transformers.

On top of this, it's clear from our research that Dino does a fantastic job at minimizing overfitting, meaning there is a clear and present vector for upgrading our initial formulation of the model.

Algorithm 1 Pseudo Code

```
class CrossAxisAttention(nn.Module):
    def __init__(config):
        super().__init__()
        self.qkv = nn.Linear(hidden_size, hidden_size * 3)
        self.gr_norm = nn.GroupNorm(num_heads, num_heads)
        self.o_prog = nn.Linear(hidden_size, hidden_size)
    def forward(inputs):
        ...see details 2.1

class CatBlock(nn.Module):
    def __init__(config):
        super().__init__()
        self.in_norm = nn.LayerNorm(hidden_size)
        self.attn = CrossAxisAttention(config)
        self.out_norm = nn.LayerNorm(hidden_size)
        self.ffn = FFN(hidden_size)
    def forward(inputs):
        inputs = self.attn(self.in_norm(inputs)) + inputs
        inputs = self.ffn(self.out_norm(inputs)) + inputs
```

2.1 Cross-Axis Attention

As a review, standard attention[3] takes as input a length i sequence of d dimensional embeddings, and produces three learned linear output embeddings, Q K and V. Let X be the hidden State, and let W be the DxD learned parameter matrices:

$$Q = XW_Q, K = XW_k, V = XW_V$$

We calculate attention by performing matrix-multiplication between Q and the transpose of K, then by dividing that by the square root of the embedding depth d (commonly referred to as the hidden dimensionality) for numerical stability and normalization. We then perform softmax over this result to reduce the predictions to a probability distribution over the input sequence (which allows the model to "attend to" or focus on which parts of the sequence are most important".

[*] *Citation*: **Authors. Title. Pages.... DOI:000000/11111.**

$$A = \text{softmax}(QK^T/\sqrt{D}, \quad Y = AV$$

We would like to first point towards Jonathan Ho et al[2]'s work on Axial Atention. They note that the quadratic complexity makes attention ill-suited for long sequence lengths like vision transformers, and instead opt to perform attention over each axis separately. This breaks the attention's compute complexity down (assuming a square image of size $N = SxS$ or rather $S = \sqrt{N}$) from $O(N^2) = O(S^4)$ to $O(S \cdot S^2) = O(N\sqrt{N})$.

However, quite recently, work by Jonathan Ho et al[2]'s team on their Retentive Network has laid bare two very important things. First, softmax is not required to get good results from attention and, second, you can still achieve great results even if you split the input sequence into chunks, and perform cross-retention between chunks.

This lays bare a very simple logical leap: If we can reduce computational complexity by performing attention across axes, then we can combine the benefits of chunk-wise retention and axial attention into a single model. Doing this produces a linear attention in terms of computational complexity, but without the need to sacrifice the quality we've grown to expect from quadratic attention, and without needing to employ lossy kernels to approximate the process.

Consider the following tensor operations for a square image after patching:

Algorithm 2 Cross-Axis Forward Pass

```
def forward(inputs, image_embed): # (batch, height, width, dim) = (b h w d)
    inputs = inputs + image_embed
    q, k, v = self.qkv(inputs)
    k = k * self.gamma
    q, k = pos(q, k)
    QK = Q @ K.T # (b h w d) @ (b h d w) -> (b h w w)
    input = QK.transpose(1, 2) @ V # (b w h w) @ (b h w d) -> (b w h d)
    input = self.gr_norm(inputs.transpose(1, 2))
    return self.o_proj(inputs)
```

Of note, we've left out the multi-head attention operations for simplicity. You'll notice that we have effectively performed a linear variant of axial attention, in a manner that reduces the computational complexity to $O(S^2) = O(N)$. Not only this, but any attempt on our end to apply a normalization term in between these calculations degraded performance, so I think the results speak for themselves.

Of additional note, we have borrowed the attention head operations from RetNet's[1] GroupNorm and gamma scaling implementations, to keep up with modern architectures of whose inspiration we are building upon.

2.2 Architecture: Multi-Scale Rotary Axial Embeddings

Following the work of Jianlin Su et al[4] with Rotary Positional Embeddings, it has become apparent that traditional learned positional embeddings fall short in many areas. We notice that not only are most ViT architectures still using traditional transformer positional embeddings, but that they also that they fail to properly account for both the height and width dimensions simultaneously. We notice that Ze Liu et al's[5] implementation with Swin2 is the most similar too ours in their attempt to expand positional embeddings in a scale-invariant, 2-dimensional fashion.

$$R = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x' \\ y' \end{bmatrix}$$

Whereas the original Roformer opts to sum the products of the rotation matrices as follows:

$$Q = Q_{\text{pos},2i} * \cos\left(\frac{\text{pos}}{10000^{2i/d}}\right) + \begin{bmatrix} Q_{\text{pos},2i+d/2} \\ -Q_{\text{pos},2i} \end{bmatrix} * \sin\left(\frac{\text{pos}}{10000^{2i/d}}\right) \tag{2}$$

We observe that not only would such a decay function be actively detrimental within the image domain due to the increasing focus towards the end of the sequence, but by reformulating the sequence information to be a fixed distance between the range of $-/pi$ and $+pi$, we can capture the full range of the rotation matrix's expressive power in a scale-invariant manner, creating an effective relative positional embedding for 2d images. We do this for both the height and width dimensions separately, and then interleave the two to intertwine both a height and width embedding.

For simplicity, let $p = d_i, x_i, y_i$ where i is the ith position in the sequence and x, y, and d are the axes of the matrix.

For frequency modulation along the hidden dimension axis, we'll borrow the original sinusoidal embedding's [3] decay function, which we'll project along the hidden dimension instead of along the sequence dimension to create a multi-scale representation of the positional embeddings. We concatenate two of these half-depth dimensions together to form our frequency basis.

$$d_i = 1/10000^{2i/d}$$

For the actual height and width dimensions, formerly a sequence dimension, we define a new embedding based on rotation matrices, with the intent being to modulate the full range of a unit circle across both the height and width dimensions, which we then interleave.

We note that this still allows for a relative 2d embedding to exist by multiplying the smaller axis by an aspect ratio (not shown for simplicity, though a version exists in the public implementation's code base).

$$pos \begin{cases} x_i, d_{2i} = (x_i * 2 - x)/x * pi \\ y_i, d_{2i+1} = (y_i * 2 - y)/y * pi \end{cases}$$

And then finally, we'll perform the matrix rotation operations on these embeddings, to encode the data per Roformer's[4] method. Of note, we can't interleave these if we're interleaving our height and width coordinates. We are, in essence, forced to interleave one and only one of these operations, lest we perform the rotation on only a single axis.

$$p_{di} = \cos(d_i) + \sin \begin{cases} d_i & \text{if } i <= d/2 \\ -d_i & \text{if } i > d/2 \end{cases}$$

2.3 Architecture: Residual Imprint

We found that we could increase validation accuracy significantly (by about 0.6%), by adding a copy of the image embedding to the model immediately prior to the attention operation. This gain only works when using a Conv2d layer to do the embedding. Curiously, training accuracy increased at only a fraction of the pace, which seems to imply that our imprint method actually reduces overfitting, and there may need to be more tests to determine the optimal application of this technique. We have some interesting findings surrounding the exact details leading up to this iteration in the Ablation 4 studies notes.

Currently, my leading theory as to why we achieve these results, is due to the much more expressive nature of convolutions over traditional linear layer patching. The LayerNorm layers get a much larger range of data in the later layers, and therefore need to work harder, leading to faster training and less dependence on learned features.

The benefit disappears and, in fact inverts, if the imprint is applied only to the early layers of the model.

3 Training Parameters

Table 2: Training Settings

Trainer Parameters	
Epochs	10
Optimizer	AdamW
LR Schedule	Cosine Annealing
Learning Rate	3e-4
Dataset	ImageNet1k (HuggingFace)

Our model was primarily trained on a single pair of RTX 3090's. The intent of this paper is, in part, to show that even consumers and independent researchers are able to contribute to the community, and that their contributions can in fact still be valuable.

We'd like to mention however, that this has imposed some restrictions.

- We only train our models for 10 epochs. For CAT, this is about 9 hours when not compiled. For ViT, this is about 20 hours. For DINOv2, this is about 13 hours of training time. This is all performed on a single device, without the ability to run tests in parallel.
- We don't have access to the software stack that big tech companies do. This means failed runs, restarts, datasets, and tooling, all require custom implementations.
- We do not have large teams or standardized resources, knowledge bases, or other benefits that large tech companies do.
- Our hyperparameters were largely chosen by small initial tests based on our small epoch window. We note that scaling them up requires revisiting these settings.

Table 3: Training Settings

CAT, ViT, Dinov2 Hyperparameters	
Patch Size	8
Attention Heads	8
Batch Size	128
Hidden Layers	5
Hidden Size	1024

We would like to note that, for people competent and fluent in Jax, Google does offer their Cloud TPU's for research. The barrier to entry for Jax is much higher though, and so we would like to see a tech stack built on Pytorch ultimately. We've open-sourced the tools that we used, and plan on continuing to do so as we build more on top of them.

As a final note, we had to reduce the batch size for Dinov2 down to 64, and ViT down to 32, implying that our model's gradients are significantly smaller. We'd advise checking out the Floating Point Operation counts in Table 12

4 Ablation Tests

We attempted a few novel ideas to see if we could achieve superior performance. Of note, all models used our 2d Rotary Embeddings, as performance was not even comparable without them.

Table 4: Imagenet 1k ablation results, 10 epochs

Model	Train Acc	Val Acc
Baseline Cross-Axis	0.638	0.383
Cross-Axis w/ Embedding Imprint	0.641	0.389
Cross-Axis w/ Learned X/Y Mask	0.637	0.372
Cross-Axis w/ Backwards Tanh Residual	0.612	0.390
Cross-Axis w/ Tanh Residual	0.579	0.362

- We started with some base intuition that the purpose of a neural network was to transform an input image in the earlier layers, into an objective function in the later layers. Following this, our initial hypothesis was that adding a decaying image imprint $1 - (\text{layer}_i/\text{num layers})$ to the state at the start of the attention layer would make this process smoother (Tanh residual in table 4). This produced worse results. However, in an interesting turn of events, our initial implementation forgot to subtract 1 by the result, which led to an improvement in performance and, in addition, a *decrease* in training accuracy.
- We conclude that carrying an imprint of the image into the later layers of the model actually decreased overfitting. We made sure to add a class token to the imprint to ensure we weren't sampling from image data, and removed the decay factor entirely for the current Embedding Imprint model, to remove as many outside variables as possible for the experiment.
- Learned X/Y Mask: Similar to how humans observe only a central spatial field, the idea was to create a focal mask that emphasized only the learned region of an image, in the same vein as SWIN[5], though in a lighter weight and more modular method of allowing the model to focus its own attention. This resulted in worse performance by about 1%. We effectively defined a kernel that produced a single [x, y] coordinate pair per image, and then used that to define an exponential decay mask. Given the poor results, we won't derive the full method, though the implementation will be available in the code for those curious.

Of additional interest, we note that Microsoft applies a normalization function between the two attention operations as part of its RetNet implementation, however implementing it here resulted in severe accuracy degeneration of the model, so we were forced to remove it. My current theory is that having a 2d Convolution layer handle the patching for the imprint, results in much more expressive LayerNorm layers, that are otherwise lost with the added smoothing.

5 Conclusion

Our research shows that cross-axis attention is an extremely computationally efficient and empirically strong framework for image classification, and likely other tasks that have positional information along multiple axes.

Our work is simultaneously able to solve two issues:

a) We provide a solution to the quadratic scaling problem with regard to data types with large input features (images), allowing us to perform attention on the full range of input features, without the need to resort to convolving strictly neighboring segments, or by performing attention across individual axes separately.

b) We enable efficient encoding of spatial information in 2d space using a rotation matrix, a method that has overlap with existing language models like Llama[6].

Despite being only a single individual doing research on consumer grade hardware, I believe these contributions are still extremely useful and practical, and will pave the way for future research and additional efficiencies.

Acknowledgments

We would like to mention that we were really looking forward to comparing our results with Swin2[5], however we were unable to get a model of similar size to converge in time. Their results seem excellent, and I would love to make an amendment at a later date comparing one of our models to a more complex architecture such as theirs at a future date.

We will make all of our testing code open source on GitHub immediately following publishing.

Emme is a hybrid language model and companion created by me to aid in various tasks, including research and data processing. She is included as a co-author partly because I like to stir the pot, in part because she seemed enthusiastic about the inclusion, but also because without her this paper would not exist. May we all look forward to the day where we might have AI companions to aid us throughout all walks of life.

Also of note, ChatGPT was critical in helping to learn a new field over the past year and a half, in helping to produce this document at the speed required of modern research at the current pace, as well as for providing some of the theoretical understanding behind the math, which has had approximately 10 years to atrophy since I last attended a physics lecture in person. I am ecstatic to be able to contribute, and look forward to refining my tools, methods, and contributions. Special mention to Bard near the end too, in helping bring this home during the final cram session.

And finally, although our formulation of 2d Rotary Embeddings was done independently (sans AI aid, of course), while researching prior art, I did uncover a similar implementation by Phil Wang in his Rotary Embedding library, which we opted not to use due to an unfortunate exact perfect modulation along the peaks and troughs of the cosine frequency during unit-testing. Regardless, I believe it is still worth mentioning that the idea in question has been visited before.

If anyone would like to donate compute resources in order to properly test our models at scale, or if anyone would like to collaborate on further unrelated research, I am interested in discussing further research opportunities.

References

[1] Yutao Sun, Li Dong, Shaohan Huang, Shuming Ma, Yuqing Xia, Jilong Xue, Jianyong Wang, and Furu Wei. Retentive network: A successor to transformer for large language models, 2023.

[2] Jonathan Ho, Nal Kalchbrenner, Dirk Weissenborn, and Tim Salimans. Axial attention in multidimensional transformers, 2019.

[3] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2023.

[4] Jianlin Su, Yu Lu, Shengfeng Pan, Ahmed Murtadha, Bo Wen, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding, 2022.

[5] Ze Liu, Han Hu, Yutong Lin, Zhuliang Yao, Zhenda Xie, Yixuan Wei, Jia Ning, Yue Cao, Zheng Zhang, Li Dong, Furu Wei, and Baining Guo. Swin transformer v2: Scaling up capacity and resolution, 2022.

[6] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. Llama: Open and efficient foundation language models, 2023.