Sequencer: Deep LSTM for Image Classification

Yuki Tatsunami[1,2] **Masato Taki**[1]
[1]Rikkyo University, Tokyo, Japan
[2]AnyTech Co., Ltd., Tokyo, Japan
{y.tatsunami, taki_m}@rikkyo.ac.jp

Abstract

In recent computer vision research, the advent of the Vision Transformer (ViT) has rapidly revolutionized various architectural design efforts: ViT achieved state-of-the-art image classification performance using self-attention found in natural language processing, and MLP-Mixer achieved competitive performance using simple multi-layer perceptrons. In contrast, several studies have also suggested that carefully redesigned convolutional neural networks (CNNs) can achieve advanced performance comparable to ViT without resorting to these new ideas. Against this background, there is growing interest in what inductive bias is suitable for computer vision. Here we propose *Sequencer*, a novel and competitive architecture alternative to ViT that provides a new perspective on these issues. Unlike ViTs, Sequencer models long-range dependencies using LSTMs rather than self-attention layers. We also propose a two-dimensional version of Sequencer module, where an LSTM is decomposed into vertical and horizontal LSTMs to enhance performance. Despite its simplicity, several experiments demonstrate that Sequencer performs impressively well: Sequencer2D-L, with 54M parameters, realizes 84.6% top-1 accuracy on only ImageNet-1K. Not only that, we show that it has good transferability and the robust resolution adaptability on double resolution-band. Our source code is available at `https://github.com/okojoalg/sequencer`.

1 Introduction

The de-facto standard for computer vision has been convolutional neural networks (CNNs) [39, 64, 22, 65, 66, 9, 29, 67]. However, inspired by the many breakthroughs in natural language processing (NLP) achieved by Transformers [75, 35, 57], applications of Transformers for computer vision are now being actively studied. In particular, Vision Transformer (ViT) [16] is a pure Transformer applied to image recognition and achieves performance competitive with CNNs. Various studies triggered by ViT have shown that the state-of-the-art (SOTA) performance can be achieved for a wide range of vision tasks using self-attention alone [79, 48, 73, 47, 15], without convolution.

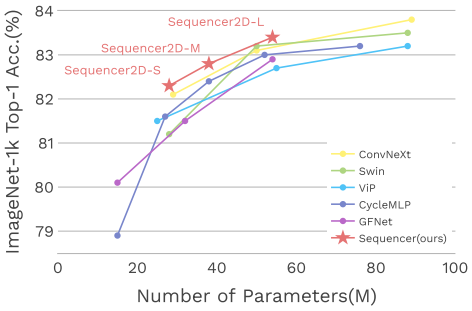


Figure 1: **IN-1K top-1 accuracy v.s. model parameters.** All models are trained on IN-1K at resolution 224^2 from scratch.

The reason for this success is thought to be due to the ability of self-attention to model long-range dependencies. However, it is still unclear how essential the self-attention is to the effectiveness of Transformers for vision tasks. Indeed, the MLP-Mixer [70] based only on multi-layer perceptrons (MLPs) is proposed as an appealing alternative to Vision Trans-

36th Conference on Neural Information Processing Systems (NeurIPS 2022).

formers (ViTs). In addition, some studies [49, 14] have shown that carefully designed CNNs are still competitive enough with Transformers in computer vision. Therefore, identifying which architectural designs are inherently effective for computer vision tasks is of great interest for current research [83]. This paper provides a new perspective on this issue by proposing a novel and competitive alternative to these vision architectures.

We propose the *Sequencer* architecture, that uses the long short-term memory (LSTM) [27] rather than the self-attention for sequence modeling. The macro-architecture design of Sequencer follows ViTs, which iteratively applies token mixing and channel mixing, but the self-attention layer is replaced by one based on LSTMs. In particular, Sequencer uses bidirectional LSTM (BiLSTM) [63] as a building block. While simple BiLSTM shows a certain level of performance, Sequencer can be further improved by using ideas similar to Vision Permutator (ViP) [28]. The key idea in ViP is to process the vertical and horizontal axes in parallel. We also introduce two BiLSTMs for top/bottom and left/right directions in parallel. This modification improves the efficiency and accuracy of Sequencer because this structure reduces the length of the sequence and yields a spatially meaningful receptive field.

When pre-trained on ImageNet-1K (IN-1K) dataset, our new attention-free architecture outperforms advanced architectures such as Swin [48] and ConvNeXt [49] of comparable size, see Figure 1. It also outperforms other attention-free and CNN-free architectures such as MLP-Mixer [70] and GFNet [61], making Sequencer an attractive new alternative to the self-attention mechanism in vision tasks.

This study also aims to propose novel architecture with practicality by employing LSTM for spatial pattern processing. Notably, Sequencer exhibits robust resolution adaptability, which strongly prevents accuracy degradation even when the input's resolution is increased double during inference. Moreover, fine-tuning Sequencer on high-resolution data can achieve higher accuracy than Swin-B [48] and Sequencer is also useful for semantic segmentation. On peak memory, Sequencer tends to be more economical than ViTs and recent CNNs for high-resolution input. Although Sequencer requires more FLOPs than other models due to recursion, the higher resolution improves the relative efficiency of peak memory, enhancing the accuracy/cost trade-off at a high-resolution regime. Therefore, Sequencer also has attractive properties as a practical image recognition model.

2 Related works

Inspired by the success of Transformers in NLP [75, 35, 57, 58, 3, 60], various applications of self-attention have been studied in computer vision. For example, in iGPT [6], an attempt was made to apply autoregressive pre-training with causal self-attention [57] to image classification. However, due to the computational cost of pixel-wise attention, it could only be applied to low-resolution images, and its ImageNet classification performance was significantly inferior to the SOTA. ViT [16], on the other hand, quickly brought Transformer's image classification performance closer to SOTA with its idea of applying bidirectional self-attention [35] to image patches rather than pixels. Various architectural and training improvements [72, 84, 79, 90, 48, 73, 5] have been attempted for ViT [16]. In this paper, we do not improve self-attention itself but propose a completely new module for image classification to replace it.

The extent to which attention-based cross-token communication inherently contributes to ViT's success is not yet well understood, starting with MLP-Mixer [70], which completely replaced ViT's self-attention with MLP, various MLP-based architectures [71, 46, 28, 69, 68, 13] have achieved competitive performance on the ImageNet dataset. We refer to these architectures as global MLPs (GMLPs) because they have global receptive fields. This series of studies cast doubt on the need for self-attention. From a practical standpoint, however, these MLP-based models have a drawback: they need to be finetuned to cope with flexible input sizes during inference by modifying the shape of their token-mixing MLP blocks. This resolution adaptability problem has been improved in CycleMLP [7], for example, by the idea of realizing a local kernel with a cyclic MLP. There are similar ideas such as [82, 81, 42, 21] which are collectively referred to as local MLPs (LMLPs). Besides the MLP-based idea, several other interesting self-attention alternatives have been found. GFNet [61] uses Fourier transformation of the tokens and mixes the tokens by global filtering in the frequency domain. PoolFormer [83], on the other hand, achieved competitive performance with only local pooling of tokens, demonstrating that simple local operations are also a suitable alternative. Our proposed Sequencer is a new alternative to self-attention that differs from both of the above, and

Sequencer is an attempt to realize token mixing in vision architectures using only LSTM. It achieved competitive performance with SOTA on the IN-1K benchmark, especially with an architecture that can flexibly adapt to higher resolution.

The idea of spatial axis decomposition has been used several times in neural architecture in computer vision. For example, SqueezeNeXt [17] decomposes a 3x3 convolution layer into 1x3 and 3x1 convolution layers, resulting in a lightweight model. Criss-cross attention [31] reduces memory usage and computational complexity by restricting the attention to only vertical and horizontal portions. Current architectures such as CSwin [15], Couplformer [40], ViP [28], RaftMLP [69], SparseMLP [68], and MorphMLP [86] have included similar ideas to improve efficiency and performance.

In the early days of deep learning, there were attempts to use RNNs for image recognition. The earliest study that applied RNNs to image recognition is [19]. The primary difference between our study and [19] is that we utilize a usual RNN in place of a 2-multi-dimensional RNN(2MDRNN). The 2MDRNN requires $H + W$ sequential operations; The LSTM requires H sequential operations, where H and W are height and width, respectively. For subsequent work on image recognition using 2MDRNNs, see [20, 32, 4, 43]. [4] proposed an architecture in which information is collected from four directions (upper left, lower left, upper right, and lower right) by RNNs for understanding natural scene images. [43] proposed a novel 2MDRNN for semantic object parsing that integrates global and local context information, called LG-LSTM. The overall architecture design is structured to input deep ConvNet features into the LG-LSTM, unlike Sequencer which stacks LSTMs. ReNet [77] is most relevant to our work; ReNet [77] uses a 4-way LSTM and non-overlapping patches as input. In this respect, it is similar to Sequencer. Meanwhile, there are three differences. First, Sequencer is the first MetaFormer [83] realized by adopting LSTM as the token mixing block. Sequencer also adopts a larger patch size than ReNet [77]. The benefit of adopting these designs is that we can modernize LSTM-based vision architectures and fairly compare LSTM-based models with ViT. As a result, our results provide further evidence for the extremely interesting hypothesis MetaFormer [83]. Second, the way vertical BiLSTMs and horizontal BiLSTMs are connected is different. Our work connects them in parallel, allowing us to gather vertical and horizontal information simultaneously. On the other hand, in ReNet [77], the output of the horizontal BiLSTM is used as input to the vertical BiLSTM. Finally, we trained Sequencer on large datasets such as ImageNet, whereas ReNet [77] is limited to small datasets as MNIST [41], CIFAR-10 [38], and SVHN [54], and has not shown the effectiveness of LSTM for larger datasets. ReSeg [76] applied ReNet to semantic segmentation. RNNs have been applied not only to image recognition, but also to generative models: PixcelRNN [74] is a pixel-channel autoregressive generative model of images using Row RNN, which consists of a 1D-convolution and a usual RNN, and Diagonal BiLSTM, which is computationally expensive.

In NLP, attempts have been made to avoid the computational cost of attention by approximating causal self-attention with recurrent neural network (RNN) [34] or replacing it with RNN after training [33]. In particular, in [34], an autoregressive pixel-wise image generation task is experimented with an architecture where the attentions in iGPT are approximated by RNNs. These studies are specific to unidirectional Transformers, in contrast to our token-based Sequencer which is the bidirectional analog of them.

3 Method

In this section, we briefly recap the preliminary background on LSTM and further describe the details of the proposed architectures.

3.1 Preliminaries: Long short-term memory

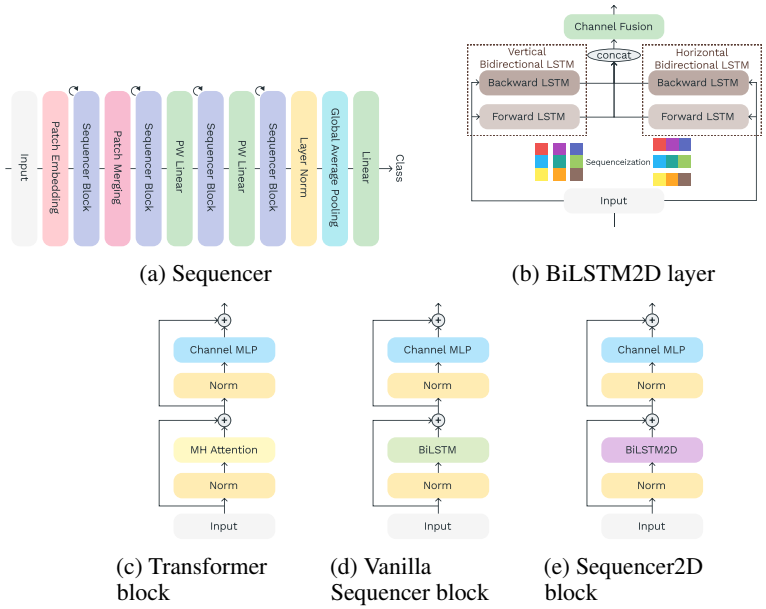
LSTM [27] is a specialized recurrent neural network (RNN) for modeling long-term dependencies of sequences. Plain LSTM has an input gate i_t that controls the storage of inputs, a forget gate f_t that controls the forgetting of the former cell state c_{t-1} and an output gate o_t that controls the cell output h_t from the current cell state c_t. Plain LSTM is formulated as follows:

$$i_t = \sigma\left(W_{xi}x_t + W_{hi}h_{t-1} + b_i\right), \quad f_t = \sigma\left(W_{xf}x_t + W_{hf}h_{t-1} + b_f\right), \tag{1}$$

$$c_t = f_t \odot c_{t-1} + i_t \odot \tanh\left(W_{xc}x_t + W_{hc}h_{t-1} + b_c\right), \quad o_t = \sigma\left(W_{xo}x_t + W_{ho}h_{t-1} + b_o\right), \tag{2}$$

$$h_t = o_t \odot \tanh(c_t), \tag{3}$$

where σ is the logistic sigmoid function and $\odot$ is Hadamard product.



(a) Sequencer (b) BiLSTM2D layer

(c) Transformer block (d) Vanilla Sequencer block (e) Sequencer2D block

Figure 2: (a) The architecture of Sequencers; (b) The figure outlines the BiLSTM2D layer, which is the main component of Sequencer2D. (c) Transformer block consist of multi-head attention. In contrast, (d) Vanilla Sequencer block and (e) Sequencer2D block, utilized on our archtecture, composed of BiLSTM or BiLSTM2D instead of multi-head attention.

BiLSTM [63] is profitable for sequences where mutual dependencies are expected. A BiLSTM consists of two plain LSTMs. Let $\overrightarrow{x}$ be the input series and $\overleftarrow{x}$ be the rearrangement of $\overrightarrow{x}$ in reverse order. $\overrightarrow{h_{\text{for}}}$ and $\overleftarrow{h_{\text{back}}}$ are the outputs obtained by processing $\overrightarrow{x}$ and $\overleftarrow{x}$ with the corresponding LSTMs, respectively. Let $\overrightarrow{h_{\text{back}}}$ be the output $\overleftarrow{h_{\text{back}}}$ rearranged in the original order, and the output of BiLSTM is obtained as follows:

$$\overrightarrow{h_{\text{for}}}, \overleftarrow{h_{\text{back}}} = \text{LSTM}_{\text{for}}(\overrightarrow{x}), \text{LSTM}_{\text{back}}(\overleftarrow{x}), \quad h = \texttt{concatenate}(\overrightarrow{h_{\text{for}}}, \overrightarrow{h_{\text{back}}}) \,. \tag{4}$$

Assume that both $\overrightarrow{h_{\text{for}}}$ and $\overrightarrow{h_{\text{back}}}$ have the same hidden dimension D, which is hyperparameter of BiLSTM. Accordingly, vector h has dimension $2D$.

3.2 Sequencer architecture

Overall architecture In the last few years, ViT and its many variants based on self-attention [16, 72, 48, 91] have attracted much attention in computer vision. Following these, several works [70, 71, 46, 28] have been proposed to replace self-attention with MLP. There have also been studies of replacing self-attention with a hard local induced bias module [7, 83] and with a global filter [61] using the fast Fourier transform algorithm (FFT) [10]. This paper continues this trend and attempts to replace the self-attention layer with LSTM [27]: we propose a new architecture aiming at memory saving by mixing spatial information with LSTM, which is memory-economical compared to ViT, parameter-saving, and has the ability to learn long-range dependencies.

Figure 2a shows the overall structure of Sequencer architecture. Sequencer architecture takes non-overlapping patches as input and projects them onto the feature map. Sequencer block, which is a core component of Sequencer, consists of the following sub-components: (1) BiLSTM layer can mix spatial information more memory-economically for high-resolution images than Transformer layer and more globally than CNN. (2) MLP for channel-mixing as well as [16, 70]. Sequencer block is called Vanilla Sequencer block when plain BiLSTM layers are used as BiLSTM layers as Figure 2d and Sequencer2D block when BiLSTM2D layers are used as Figure 2e. We define BiLSTM2D layer later. The output of the last block is sent to the linear classifier via the global average pooling layer, as in most other architectures.

BiLSTM2D layer We propose the BiLSTM2D layer as a technique to mix 2D spatial information efficaciously. It has two plain BiLSTMs: a vertical BiLSTM and a horizontal one. For an input $\mathbf{X} \in \mathbb{R}^{H \times W \times C}$, $\{\mathbf{X}_{:,w,:} \in \mathbb{R}^{H \times C}\}_{w=1}^{W}$ is viewed as a set of sequences, where H is the number of tokens in the vertical direction, W is the number of sequences in the horizontal direction, and C is the channel dimension. All sequences $\mathbf{X}_{:,w,:}$ are input into the vertical BiLSTM with shared weights and hidden dimension D:

$$\mathbf{H}^{\text{ver}}_{:,w,:} = \text{BiLSTM}(\mathbf{X}_{:,w,:}). \tag{5}$$

In a very similar manner, $\{\mathbf{X}_{h,:,:} \in \mathbb{R}^{W \times C}\}_{h=1}^{H}$ is viewed as a set of sequences, and all sequences $\mathbf{X}_{h,:,:}$ are input into the horizontal BiLSTM with shared weights and hidden dimension D as well:

$$\mathbf{H}^{\text{hor}}_{h,:,:} = \text{BiLSTM}(\mathbf{X}_{h,:,:}). \tag{6}$$

We combine $\{\mathbf{H}^{\text{ver}}_{:,w,:} \in \mathbb{R}^{H \times 2D}\}_{w=1}^{W}$ into $\mathbf{H}^{\text{ver}} \in \mathbb{R}^{W \times H \times 2D}$ and $\{\mathbf{H}^{\text{hor}}_{h,:,:} \in \mathbb{R}^{W \times 2D}\}_{h=1}^{H}$ into $\mathbf{H}^{\text{hor}} \in \mathbb{R}^{W \times H \times 2D}$. They are then concatenated and processed point-wisely in a fully-connection layer. These processes are formulated as follows:

$$\mathbf{H} = \texttt{concatenate}(\mathbf{H}^{\text{ver}}, \mathbf{H}^{\text{hor}}), \quad \hat{\mathbf{X}} = \text{FC}(\mathbf{H}), \tag{7}$$

where $\text{FC}(\cdot)$ denotes the fully-connected layer with weight $W \in \mathbb{R}^{C \times 4D}$. The `PyTorch`-like pseudocode is shown in Appendix B.1.

BiLSTM2D is more memory-economical and throughput-efficiency than multi-head-attention of ViT for high-resolution input. BiLSTM2D involves $(WC + HC)/2$ dimensional cell states, while a multi-head-attention involves $h * (HW)^2$ dimensional attention map where h is a number of heads. Thus, as H and W increase, the memory cost of an attention map increases more rapidly than the cost of a cell state. On throughput, the computational complexity of self-attention is $\mathcal{O}(W^4C)$, whereas the computational complexity of BiLSTM is $\mathcal{O}(WC^2)$ where we assume $W = H$ for simplicity. There are $\mathcal{O}(W)$ sequential operations for BiLSTM2D. Therefore, assuming we use a sufficiently efficient LSTM cell implementation, such as official PyTorch LSTMs we are using, the increase of the complexity of self-attention is much more rapid than BiLSTM2D. It implies a lower throughput of attention compared to BiLSTM2D. See an experiment in Section 4.5.

Architecture variants For comparison between models of different depths consisting of Sequencer2D blocks, we have prepared three models with different depths: 18, 24, and 36. The names of the models are *Sequencer2D-S*, *Sequencer2D-M*, and *Sequencer2D-L*, respectively. The hidden dimension is set to $D = C/4$. Details of these models are provided in Appendix B.2.

As shown in subsection 4.1, these architectures outperform typical models. Interestingly, however, subsection 4.3 shows that replacing Sequencer2D block with the simpler Vanilla Sequencer block maintains moderate accuracy. We denote such a model as Vanilla Sequencer. Note that some of the explicit positional information is lost in the *Vanilla Sequencer* because the model treats patches as a 1D sequence.

4 Experiments

In this section, we compare Sequencers with previous studies on the IN-1K benchmark [39]. We also carry out ablation studies, transfer learning studies, and analysis of the results to demonstrate the effectiveness of Sequencers. We adopt `PyTorch` [56] and `timm` [80] library to implement models in the conduct of all experiments. See Appendix B for more setup details.

4.1 Scratch training on IN-1K

We utilize IN-1K [39], which has 1000 classes and contains 1,281,167 training images and 50,000 validation images. We adopt AdamW optimizer [50]. Following the previous study [72], we adopt the base learning rate $\frac{\text{batch size}}{512} \times 5 \times 10^{-4}$. The batch sizes for Sequencer2D-S, Sequencer2D-M, and Sequencer2D-L are 2048, 1536, and 1024, respectively. As a regularization method, stochastic depth [30] and label smoothing [66] are employed. As data augmentation methods, mixup [87], cutout [12], cutmix [85], random erasing [88], and randaugment [11] are applied.

Table 1: The table shows the top-1 accuracy when trained on IN-1K, comparing our model with other similar scale representative models. Training and inference throughput and their peak memory were measured with 16 images per batch on a single V100 GPU. The left sides of the slashes are values during training, and the right sides of the slashes are values during inference. Fine-tuned models marked with "↑". Note Sequencer2D-L↑ are compared to Swin-B↑ and ConvNeXt-B↑ with more parameters since Swin and ConvNeXt have not fine-tuned models of similar parameters with Sequencer2D-L↑ in the original papers.

Model	Family	Res.	#Param.	FLOPs	Throughput (image/s)	Peak Mem. (MB)	Top-1 Acc.(%)	Pre FT Top-1 Acc.(%)
Training from scratch								
RegNetY-4GF [59]	CNN	224^2	21M	4.0G	228/823	1136/225	80.0	non-fine-tune
ConvNeXt-T [49]	CNN	224^2	29M	4.5G	337/1124	1418/248	82.1	as above
DeiT-S [72]	Trans.	224^2	22M	4.6G	480/1569	1195/180	79.9	as above
Swin-T [48]	Trans.	224^2	28M	4.5G	268/894	1613/308	81.2	as above
ViP-S/7 [28]	GMLP	224^2	25M	6.9G	214/702	1587/195	81.5	as above
CycleMLP-B2 [7]	LMLP	224^2	27M	3.9G	158/586	1357/234	81.6	as above
PoolFormer-S24 [83]	LMLP	224^2	21M	3.6G	313/988	1461/183	80.3	as above
Sequencer2D-S	Seq.	224^2	28M	8.4G	110/347	1799/196	**82.3**	as above
RegNetY-8GF [59]	CNN	224^2	39M	8.0G	211/751	1776/333	81.7	as above
T2T-ViT$_t$-19 [84]	Trans.	224^2	39M	9.8G	197/654	3520/1140	82.2	as above
CycleMLP-B3 [7]	LMLP	224^2	38M	6.9G	100/367	2326/287	82.6	as above
PoolFormer-S36 [83]	LMLP	224^2	31M	5.2G	213/673	2187/220	81.4	as above
GFNet-H-S [61]	FFT	224^2	32M	4.5G	227/755	1740/282	81.5	as above
Sequencer2D-M	Seq.	224^2	38M	11.1G	83/270	2311/244	**82.8**	as above
RegNetY-12GF [59]	CNN	224^2	46M	12.0G	199/695	2181/440	82.4	as above
ConvNeXt-S [49]	CNN	224^2	50M	8.7G	212/717	2265/341	83.1	as above
Swin-S [48]	Trans.	224^2	50M	8.7G	165/566	2635/390	83.2	as above
Mixer-B/16 [70]	GMLP	224^2	59M	12.7G	338/1011	1864/407	76.4	as above
ViP-M/7 [28]	GMLP	224^2	55M	16.3G	130/395	3095/396	82.7	as above
CycleMLP-B4 [7]	LMLP	224^2	52M	10.1G	70/259	3272/338	83.0	as above
PoolFormer-M36 [83]	LMLP	224^2	56M	9.1G	171/496	3191/368	82.1	as above
GFNet-H-B [61]	FFT	224^2	54M	8.4G	144/482	2776/367	82.9	as above
Sequencer2D-L	Seq.	224^2	54M	16.6G	54/173	3516/322	**83.4**	as above
Fine-tuning								
ConvNeXt-B↑ [49]	CNN	384^2	89M	45.1G	78/234	7329/870	85.1(+1.3)	83.8
Swin-B↑ [48]	Trans.	384^2	88M	47.1G	54/156	12933/1532	84.5(+1.0)	83.5
GFNet-B↑ [61]	FFT	384^2	47M	23.2G	137/390	3710/416	82.1(+0.8)	82.9
Sequencer2D-L↑	Seq.	392^2	54M	50.7G	26/84	9062/481	84.6(+1.2)	83.4

Table 1 shows the results that are comparing the proposed models to others with a comparable number of parameters to our models, including models with local and global receptive fields such as CNNs, ViTs, and MLP-based and FFT-based models. Sequencers has the disadvantage that its throughput is slower than other models because it uses RNNs. In the scratch training on IN-1K, however, they outperform these recent comparative models in accuracy across their parameter bands. In particular, Seqeuncer2D-L is competitive with recently discussed models with comparable parameters such as ConvNeXt-S [49] and Swin-S [48], with accuracy outperformance of 0.3% and 0.2%, respectively.

Table 1 demonstrates that Sequencer's throughput is not good. The training throughput is about three times the inference throughput for all these models. Compared to other models, both measured inference and training time are not good.

4.2 Fine-tuning on IN-1K

In this fine-tuning study, Sequencer2D-L pre-trained on IN-1K at 224^2 resolution is fine-tuned on IN-1K at 392^2 resolution. We compare it with the other models fine-tuned on IN-1K at 384^2

Table 2: **Sequencer ablation experiments**. We adopt Sequencer2D-S variant for these ablation studies. **C1** denotes vertical BiLSTM, **C2** denotes horizontal BiLSTM, and **C3** denotes channel fusion component. When vertical BiLSTM only, horizontal BiLSTM only or unidirectional BiLSTM2D, its hidden dimension needs to be doubled from the original setting because it compensates the output dimension for the excluded LSTM and matches the dimensions.

(a) Components

C1	C2	C3	Acc.
✓			75.6
	✓		75.0
✓	✓		81.6
✓	✓	✓	**82.3**

(b) LSTM Direction

Bidirectional	Acc.
	79.7
✓	**82.3**

(c) Vanilla Sequencer

Model	#Params.	FLOPs	Acc.
VSequencer-S	33M	8.4G	78.0
VSequencer(H)-S	28M	8.4G	78.8
VSequencer(PE)-S	33M	8.4G	78.1
Sequencer2D-S	28M	8.4G	**82.3**

(d) Hidden dimension

Hidden dim. ratio	#Params.	FLOPs	Acc.
1x	**28M**	**8.4G**	82.3
2x	45M	13.9G	**82.6**

(e) Various RNNs

Model	#Params.	FLOPs	Acc.
RNN-Sequencer2D	19M	5.8G	80.6
GRU-Sequencer2D	25M	7.5G	**82.3**
Seqeucer2D-S	28M	8.4G	**82.3**

resolution. Since 392 is divisible by 14, the input at this resolution can be split into patches without padding. However, note that this is not the case with a resolution of 384^2.

As Table 1 indicates, even when higher-resolution Sequencer is fine-tuned, it is competitive with the latest models such as ConvNeXt [49], Swin [48], and GFNet [61].

4.3 Ablation studies

This subsection presents ablation studies based on Sequencer2D-S for further understanding of Sequencer. We seek to clarify the effectiveness and validity of the Sequencers architecture in terms of the importance of each component, bidirectional necessaries, setting of the hidden dimension, and the comparison with simple BiLSTM.

We show where and how relevant the components of BiLSTM2D are: The BiLSTM2D is composed of vertical BiLSTM, horizontal BiLSTM, and channel fusion elements. We want to see the validity of vertical BiLSTM, horizontal BiLSTM, and channel fusion. For this purpose, we examine the removal of channel fusion and vertical or horizontal BiLSTM. Table 2a shows the results. Removing channel fusion shows that the performance degrades from 82.3% to 81.6%. Furthermore, the additional removal of vertical or horizontal BiLSTM exposes a 6.0% or 6.6% performance drop, respectively. Hence, each component discussed here is necessary for Sequencer2D.

We show that the bidirectionality for BiLSTM2D is important for Sequencer. We compare Sequencer2D-S with a version that replaces the vertical and horizontal BiLSTMs with vertical and horizontal unidirectional LSTMs. Table 2b shows that the unidirectional model is 2.6% less accurate than the bidirectional model. This result attests to the significance of using not unidirectional LSTM but BiLSTM.

It is important to set the hidden dimension of LSTM to a reasonable size. As described in subsection 3.2, Sequencer2D sets the hidden dimension D of BiLSTM to $D = C/4$, but this is not necessary if the model has channel fusions. Table 2d compares Sequencer2D-S with the model with increased D. Although accuracy is 0.3% improved, FLOPs increase by 65%, and the number of parameters increases by 60%. Namely, the accuracy has not improved for the increase in FLOPs. Moreover, the increase in dimension causes overfitting, which is discussed in Appendix C.3.

Vanilla Sequencer can also achieve accuracy that outperforms MLP-Mixer [70], but is not as accurate as Sequencer2D. Following experimental result supports the claim. We experiment with the Sequencer2D-S variants, where Vanilla Sequencer blocks replace the Sequencer2D blocks, called VSequencer-S(H), with incomplete positional information. In addition, we experiment with a variant of VSequencer-S(H) without the hierarchical structure, which we call VSequencer-S. VSequencer-S(PE) is VSequencer-S using ViTs-style learned positional embedding (PE) [16]. Table 2c indicates effectiveness for combination of LSTM and ViTs-like architecture. Surprisingly, even with Vanilla

Table 3: **Left.** Results on transfer learning. We transfer models trained on IN-1K to datasets from different domains. Sequencers use 224^2 resolution images, while ViT-B/16 and EfficientNet-B7 work on higher resolution, see Res. column. **Right.** Semantic segmentation results on ADE20K [89]. All models are Semantic FPN [36] based. We show mIoU for the ADE20k validation set.

Model	Res.	#Pr.	FLOPs	CF_{10}	CF_{100}	Flowers	Cars
ResNet50 [22]	224^2	26M	4.1G	-	-	96.2	90.0
EN-B7 [67]	600^2	26M	37.0G	98.9	91.7	98.8	94.7
ViT-B/16 [16]	384^2	86M	55.4G	98.1	87.1	89.5	-
DeiT-B [72]	224^2	86M	17.5G	99.1	90.8	98.4	92.1
CaiT-S-36 [73]	224^2	68M	13.9G	99.2	92.2	98.8	93.5
ResMLP-24 [71]	224^2	30M	6.0G	98.7	89.5	97.9	89.5
GFNet-H-B [61]	224^2	54M	8.6G	99.0	90.3	98.8	93.2
Sequencer2D-S	224^2	28M	8.4G	99.0	90.6	98.2	93.1
Sequencer2D-M	224^2	38M	11.1G	99.1	90.8	98.2	93.3
Sequencer2D-L	224^2	54M	16.6G	99.1	91.2	98.6	93.1

Model	#Pr.	mIoU
PVT-Small [79]	28M	39.8
PoolFormer-S24 [83]	23M	40.3
Sequencer2D-S	32M	46.1
PVT-Medium [79]	48M	41.6
PoolFormer-S36[83]	35M	42.0
Sequencer2D-M	42M	47.3
PVT-Large [79]	65M	42.1
PoolFormer-M36 [83]	60M	42.4
Sequencer2D-L	58M	48.6

Sequencer and Vanilla Sequencer(H) without PE, the performance reduction from Sequencer2D-S is only 4.3% and 3.5%, respectively. According to these results, there is no doubt that Vanilla Sequencer using BiLSTMs is significant enough, although not as accurate as Sequencer2D.

All LSTMs in the BiLSTM2D layer can be replaced with other recurrent networks such as gated recurrent units (GRUs) [8] or tanh-RNNs to define BiGRU2D layer or BiRNN2D layer. We also trained these models on IN-1K, so see Table 2e for the results. The table suggests that all of these variants, including RNN-cell, work well. Also, tanh-RNN performs slightly worse than others, probably due to its lower ability to model long-range dependence.

4.4 Transfer learning and semantic segmentation

Sequencers perform well on IN-1K, and they have good transferability. In other words, they have satisfactory generalization performance for a new domain, which is shown below. We utilize the commonly used CIFAR-10 [38], CIFAR-100 [38], Flowers-102 [55], and Stanford Cars [37] for this experiment. See the references and Appendix B.4 for details on the datasets. The results of the proposed model and the results in previous studies of models with comparable capacity are presented in Table 3. In particular, Sequencer2D-L achieves results that are competitive with CaiT-S-36 [73] and EfficientNet-B7 [67].

We experiment for semantic segmentation on ADE20K[89] dataset. See Appendix C.4 for details on the setup. Sequencer outperforms PVT [79] and PoolFormer [83] with similar parameters; compared to PoolFormer, mIoU is about 6 pts higher.

We have investigated a commonly object detection model with Sequencer as the backbone. Its performance is not much different from the case of ResNet [22] backbone. Its improvement is the future work. See Appendix C.5.

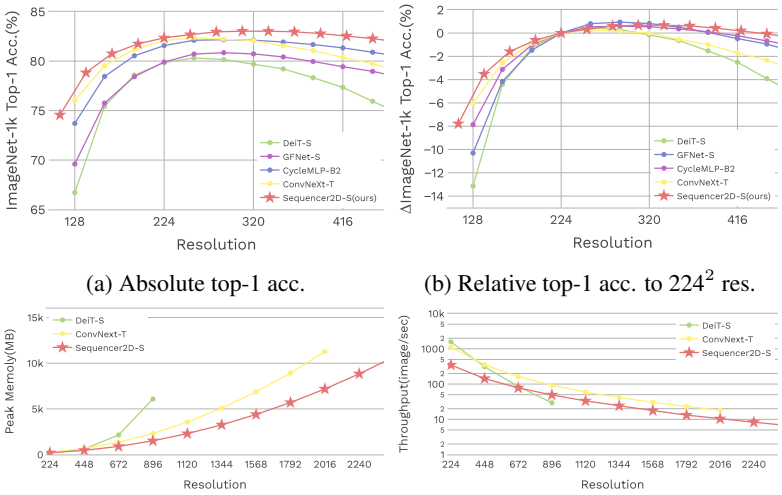
4.5 Analysis and visualization

In this subsection, we investigate the properties of Sequencer in terms of resolution adaptability and efficiency. Furthermore, effective receptive field (ERF) [51] and visualization of the hidden states provides insight into the question of how Sequencer recognizes images.

One of the attractive properties of Sequencer is its flexible adaptability to the resolution, with minimal impact on accuracy even when the resolution of the input image is varied from one-half to twice. In comparison, architectures like MLP-Mixer [70] have a fixed input resolution, and GFNet [61] requires interpolation of weights in the Fourier domain when inputting images with a resolution different from training images. We evaluate the resolution adaptability of models comparatively by inputting different resolution images to each model, without fine-tuning, with pre-trained weights on IN-1K at the resolution of 224^2. Figure 3a compares absolute top-1 accuracy on IN-1K, and Figure 3b compares relative one to the input image with the resolution of 224^2. By increasing the resolution by 28 for Sequencer2D-S and by 32 for other models, we avoid padding and prevent

the effect of padding on accuracy. Compared to DeiT-S [72], GFNet-S [61], CycleMLP-B2 [7], and ConvNeXt-T [49], Sequencer-S's performance is more sustainable. The relative accuracy is consistently better than ConvNeXt [49], which is influential in the lower-resolution band, and, at 448^2 resolution, 0.6% higher than CycleMLP [7], which is influential in the double-resolution band. It is noteworthy that Sequencer continues to maintain high accuracy on double resolution.

The higher the input resolution, the higher memory-efficiency and throughput of Sequencers when compared to DeiT [72]. Figure 3 shows the efficiency of Sequencer2D-S when compared to DeiT-S and ConvNeXt-T [49]. Memory consumption increases rapidly in DeiT-S and ConvNeXt-T with increasing input resolution, but more gradual increase in Sequencer2D-S. The result strongly implies that it has more practical potential as the resolution increases than the ViTs. At a resolution of 224^2, it is behind DeiT in throughput, but it stands ahead of DeiT when images with a resolution of 896^2 are input.



(a) Absolute top-1 acc.

(b) Relative top-1 acc. to 224^2 res.

(c) GPU peak memory

(d) Throughput

Figure 3: **Top.** Resolution adaptability. Every model is trained at 224^2 resolution and evaluated at various resolutions with no fine-tuning. **Bottom.** Comparisons among Sequencer2D-S, DeiT-S [72], and ConvNeXt-T [49] in (c) GPU peak memory and (d) throughput for different input image resolutions. Measured for each increment of 224^2 resolution, points not plotted are when GPU memory is exhausted. The measurements are founded on a batch size of 16 and a single V100.

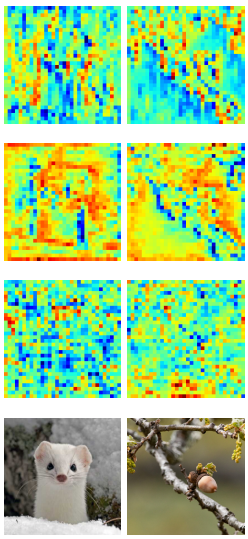
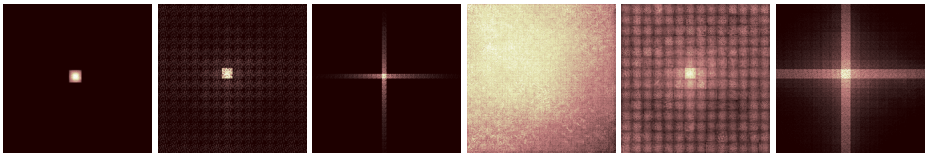


Figure 4: Part of states of the last BiLSTM2D layer in the Sequencer block of stage 1. From top to bottom: outputs of ver-LSTM, hor-LSTM, and ch-fusions and original images.

In general, CNNs have localized, layer-by-layer expanding receptive fields, and ViTs without shifted windows capture global dependencies, working the self-attention mechanism. In contrast, in the case of Sequencer, it is not clear how information is processed in Sequencer block. We calculated ERF [51] for ResNet-50 [22], DeiT-S [72], and Sequencer2D-S as shown in Figure 5. ERFs of Sequencer2D-S form a cruciform shape in all layers. The trend distinguishes it from well-known models such as DeiT-S and ResNet-50. More remarkably, in shallow layers, Sequencer2D-S has a wider ERF than ResNet-50, although not as wide as DeiT. This observation confirms that LSTMs in Sequencer can model long-term dependencies as expected and that Sequencer recognizes sufficiently long vertical or horizontal regions. Thus, it can be argued that Sequencer recognizes an image in a very different way than CNNs or ViTs. For more details on ERF and additional visualization, see Appendix D.

Moreover we also visualized a hidden state of vertical and horizontal BiLSTM, and a feature map after channel fusion, and the results are visualized in Figure 4. It demonstrates that our Sequencer has the hidden states interact with each other over the vertical and horizontal directions. The closer tokens are in position, the stronger their interaction tends to be; the farther tokens are in position, the less their interaction tends to be.



(a) RN50/first (b) DeiTS/first (c) SeqS/first (d) RN50/last (e) DeiTS/last (f) SeqS/last

Figure 5: The visualizations are the ERFs of Sequencer2D-S and comparative models such as ResNet-50 and DeiT-S. The left of the slash denotes the model name, and the right of the slash denotes the location of the block of output used to generate the ERFs. The ERFs are rescaled from 0 to 1. The brighter and more influential the region is, the closer to 1, and the darker, the closer to 0.

5 Conclusions

We propose a novel and simple architecture that leverages LSTM for computer vision. It is demonstrated that new modeling with LSTM instead of the self-attention layer can achieve competitive performance with current state-of-the-art models. Our experiments show that Sequencer has a good memory-resource/accuracy and parameter/accuracy tradeoffs, comparable to the main existing methods. Despite the impact of recursion on throughput, we have demonstrated benefits over it. We believe that these results raise a number of interesting issues. Improving Sequencer's poor throughput is one example. Moreover, we expect that investigating the internal mechanisms of our model using methods other than ERF will further our understanding of how this architecture works. In addition, it would be important to analyze in more detail the features learned by Sequencer in comparison to other architectures. We hope this will lead to a better understanding of the role of various inductive biases in computer vision. Furthermore, we expect that our results trigger further study beyond the domain or research area. Especially, it would be a very interesting open question to see if such a design works with time-series data in vision such as video or in a multi-modal problem setting combined with another modality such as video with audio.

Acknowledgments and Disclosure of Funding

Our colleagues at AnyTech Co., Ltd. provided valuable comments on the early versions and encouragement. We thank them for their cooperation. In particular, We thank Atsushi Fukuda for organizing discussion opportunities. We also thank people who support us, belonging to Graduate School of Artificial Intelligence and Science, Rikkyo University.

References

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

[2] Lucas Beyer, Olivier J Hénaff, Alexander Kolesnikov, Xiaohua Zhai, and Aäron van den Oord. Are we done with imagenet? *arXiv preprint arXiv:2006.07159*, 2020.

[3] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. In *NeurIPS*, volume 33, pages 1877–1901, 2020.

[4] Wonmin Byeon, Thomas M Breuel, Federico Raue, and Marcus Liwicki. Scene labeling with lstm recurrent neural networks. In *CVPR*, pages 3547–3555, 2015.

[5] Chun-Fu Richard Chen, Quanfu Fan, and Rameswar Panda. Crossvit: Cross-attention multi-scale vision transformer for image classification. In *ICCV*, pages 357–366, 2021.

[6] Mark Chen, Alec Radford, Rewon Child, Jeffrey Wu, Heewoo Jun, David Luan, and Ilya Sutskever. Generative pretraining from pixels. In *ICML*, pages 1691–1703. PMLR, 2020.

[7] Shoufa Chen, Enze Xie, Chongjian GE, Runjian Chen, Ding Liang, and Ping Luo. CycleMLP: A MLP-like architecture for dense prediction. In *ICLR*, 2022.

[8] Kyunghyun Cho, Bart van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder–decoder approaches. In *Proceedings of SSST-8, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation*, pages 103–111, 2014.

[9] François Chollet. Xception: Deep learning with depthwise separable convolutions. In *CVPR*, pages 1251–1258, 2017.

[10] James W Cooley and John W Tukey. An algorithm for the machine calculation of complex fourier series. *Mathematics of computation*, 19(90):297–301, 1965.

[11] Ekin D Cubuk, Barret Zoph, Jonathon Shlens, and Quoc V Le. RandAugment: Practical automated data augmentation with a reduced search space. In *CVPRW*, pages 702–703, 2020.

[12] Terrance DeVries and Graham W Taylor. Improved regularization of convolutional neural networks with cutout. *arXiv preprint arXiv:1708.04552*, 2017.

[13] Xiaohan Ding, Honghao Chen, Xiangyu Zhang, Jungong Han, and Guiguang Ding. Repmlpnet: Hierarchical vision mlp with re-parameterized locality. In *CVPR*, 2022.

[14] Xiaohan Ding, Xiangyu Zhang, Yizhuang Zhou, Jungong Han, Guiguang Ding, and Jian Sun. Scaling up your kernels to 31x31: Revisiting large kernel design in cnns. In *CVPR*, 2022.

[15] Xiaoyi Dong, Jianmin Bao, Dongdong Chen, Weiming Zhang, Nenghai Yu, Lu Yuan, Dong Chen, and Baining Guo. Cswin transformer: A general vision transformer backbone with cross-shaped windows. In *CVPR*, 2022.

[16] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. In *ICLR*, 2021.

[17] Amir Gholami, Kiseok Kwon, Bichen Wu, Zizheng Tai, Xiangyu Yue, Peter Jin, Sicheng Zhao, and Kurt Keutzer. Squeezenext: Hardware-aware neural network design. In *CVPRW*, pages 1638–1647, 2018.

[18] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. In *ICLR*, 2015.

[19] Alex Graves, Santiago Fernández, and Jürgen Schmidhuber. Multi-dimensional recurrent neural networks. In *International conference on artificial neural networks*, pages 549–558. Springer, 2007.

[20] Alex Graves and Jürgen Schmidhuber. Offline handwriting recognition with multidimensional recurrent neural networks. In *NeurIPS*, volume 21, 2008.

[21] Jianyuan Guo, Yehui Tang, Kai Han, Xinghao Chen, Han Wu, Chao Xu, Chang Xu, and Yunhe Wang. Hire-mlp: Vision mlp via hierarchical rearrangement. *arXiv preprint arXiv:2108.13341*, 2021.

[22] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, pages 770–778, 2016.

[23] Dan Hendrycks, Steven Basart, Norman Mu, Saurav Kadavath, Frank Wang, Evan Dorundo, Rahul Desai, Tyler Zhu, Samyak Parajuli, Mike Guo, et al. The many faces of robustness: A critical analysis of out-of-distribution generalization. In *ICCV*, pages 8340–8349, 2021.

[24] Dan Hendrycks and Thomas Dietterich. Benchmarking neural network robustness to common corruptions and perturbations. In *ICLR*, 2018.

[25] Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (GELUs). *arXiv preprint arXiv:1606.08415*, 2016.

[26] Dan Hendrycks, Kevin Zhao, Steven Basart, Jacob Steinhardt, and Dawn Song. Natural adversarial examples. In *CVPR*, pages 15262–15271, 2021.

[27] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[28] Qibin Hou, Zihang Jiang, Li Yuan, Ming-Ming Cheng, Shuicheng Yan, and Jiashi Feng. Vision permutator: A permutable mlp-like architecture for visual recognition. *IEEE TPAMI*, 2022.

[29] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *CVPR*, pages 4700–4708, 2017.

[30] Gao Huang, Yu Sun, Zhuang Liu, Daniel Sedra, and Kilian Q Weinberger. Deep networks with stochastic depth. In *ECCV*, pages 646–661, 2016.

[31] Zilong Huang, Xinggang Wang, Lichao Huang, Chang Huang, Yunchao Wei, and Wenyu Liu. Ccnet: Criss-cross attention for semantic segmentation. In *ICCV*, pages 603–612, 2019.

[32] Nal Kalchbrenner, Ivo Danihelka, and Alex Graves. Grid long short-term memory. In *In Proceedings of the IEEE Workshop on Spoken Language Technology*, 2015.

[33] Jungo Kasai, Hao Peng, Yizhe Zhang, Dani Yogatama, Gabriel Ilharco, Nikolaos Pappas, Yi Mao, Weizhu Chen, and Noah A Smith. Finetuning pretrained transformers into rnns. In *EMNLP*, pages 10630–10643, 2021.

[34] Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. Transformers are rnns: Fast autoregressive transformers with linear attention. In *ICML*, pages 5156–5165. PMLR, 2020.

[35] Jacob Devlin Ming-Wei Chang Kenton and Lee Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *NAACL-HLT*, pages 4171–4186, 2019.

[36] Alexander Kirillov, Ross Girshick, Kaiming He, and Piotr Dollár. Panoptic feature pyramid networks. In *CVPR*, pages 6399–6408, 2019.

[37] Jonathan Krause, Michael Stark, Jia Deng, and Li Fei-Fei. 3D object representations for fine-grained categorization. In *ICCVW*, pages 554–561, 2013.

[38] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. Technical report, University of Toronto, 2009.

[39] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *NeurIPS*, volume 25, pages 1097–1105, 2012.

[40] Hai Lan, Xihao Wang, and Xian Wei. Couplformer: Rethinking vision transformer with coupling attention map. *arXiv preprint arXiv:2112.05425*, 2021.

[41] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[42] Dongze Lian, Zehao Yu, Xing Sun, and Shenghua Gao. As-mlp: An axial shifted mlp architecture for vision. In *ICLR*, 2022.

[43] Xiaodan Liang, Xiaohui Shen, Donglai Xiang, Jiashi Feng, Liang Lin, and Shuicheng Yan. Semantic object parsing with local-global long short-term memory. In *CVPR*, pages 3185–3193, 2016.

[44] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. In *ICCV*, pages 2980–2988, 2017.

[45] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *ECCV*, pages 740–755, 2014.

[46] Hanxiao Liu, Zihang Dai, David So, and Quoc Le. Pay attention to mlps. In *NeurIPS*, volume 34, 2021.

[47] Ze Liu, Han Hu, Yutong Lin, Zhuliang Yao, Zhenda Xie, Yixuan Wei, Jia Ning, Yue Cao, Zheng Zhang, Li Dong, et al. Swin transformer v2: Scaling up capacity and resolution. In *CVPR*, 2022.

[48] 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 *ICCV*, 2021.

[49] Zhuang Liu, Hanzi Mao, Chao-Yuan Wu, Christoph Feichtenhofer, Trevor Darrell, and Saining Xie. A convnet for the 2020s. In *CVPR*, 2022.

[50] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *ICLR*, 2019.

[51] Wenjie Luo, Yujia Li, Raquel Urtasun, and Richard Zemel. Understanding the effective receptive field in deep convolutional neural networks. In *NeurIPS*, volume 29, 2016.

[52] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. In *ICLR*, 2018.

[53] Xiaofeng Mao, Gege Qi, Yuefeng Chen, Xiaodan Li, Ranjie Duan, Shaokai Ye, Yuan He, and Hui Xue. Towards robust vision transformer. *arXiv preprint arXiv:2105.07926*, 2021.

[54] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng. Reading digits in natural images with unsupervised feature learning. In *NeurIPS Workshop*, 2011.

[55] Maria-Elena Nilsback and Andrew Zisserman. Automated flower classification over a large number of classes. In *ICVGIP*, pages 722–729, 2008.

[56] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. In *NeurIPS*, volume 32, 2019.

[57] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding by generative pre-training. *Technical report, OpenAI*, 2018.

[58] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

[59] Ilija Radosavovic, Raj Prateek Kosaraju, Ross Girshick, Kaiming He, and Piotr Dollár. Designing network design spaces. In *CVPR*, pages 10428–10436, 2020.

[60] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *JMLR*, 21:1–67, 2020.

[61] Yongming Rao, Wenliang Zhao, Zheng Zhu, Jiwen Lu, and Jie Zhou. Global filter networks for image classification. In *NeurIPS*, volume 34, 2021.

[62] Benjamin Recht, Rebecca Roelofs, Ludwig Schmidt, and Vaishaal Shankar. Do imagenet classifiers generalize to imagenet? In *ICML*, pages 5389–5400. PMLR, 2019.

[63] Mike Schuster and Kuldip K Paliwal. Bidirectional recurrent neural networks. *IEEE TSP*, 45(11):2673–2681, 1997.

[64] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *ICLR*, 2015.

[65] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *CVPR*, pages 1–9, 2015.

[66] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *CVPR*, pages 2818–2826, 2016.

[67] Mingxing Tan and Quoc Le. EfficientNet: Rethinking model scaling for convolutional neural networks. In *ICML*, pages 6105–6114, 2019.

[68] Chuanxin Tang, Yucheng Zhao, Guangting Wang, Chong Luo, Wenxuan Xie, and Wenjun Zeng. Sparse mlp for image recognition: Is self-attention really necessary? *arXiv preprint arXiv:2109.05422*, 2021.

[69] Yuki Tatsunami and Masato Taki. Raftmlp: How much can be done without attention and with less spatial locality? *arXiv preprint arXiv:2108.04384*, 2021.

[70] Ilya O Tolstikhin, Neil Houlsby, Alexander Kolesnikov, Lucas Beyer, Xiaohua Zhai, Thomas Unterthiner, Jessica Yung, Andreas Steiner, Daniel Keysers, Jakob Uszkoreit, et al. Mlp-mixer: An all-mlp architecture for vision. In *NeurIPS*, volume 34, 2021.

[71] Hugo Touvron, Piotr Bojanowski, Mathilde Caron, Matthieu Cord, Alaaeldin El-Nouby, Edouard Grave, Armand Joulin, Gabriel Synnaeve, Jakob Verbeek, and Hervé Jégou. Resmlp: Feedforward networks for image classification with data-efficient training. *arXiv preprint arXiv:2105.03404*, 2021.

[72] Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Hervé Jégou. Training data-efficient image transformers & distillation through attention. In *ICML*, 2021.

[73] Hugo Touvron, Matthieu Cord, Alexandre Sablayrolles, Gabriel Synnaeve, and Hervé Jégou. Going deeper with image transformers. In *ICCV*, pages 32–42, 2021.

[74] Aaron Van Oord, Nal Kalchbrenner, and Koray Kavukcuoglu. Pixel recurrent neural networks. In *International conference on machine learning*, pages 1747–1756. PMLR, 2016.

[75] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *NeurIPS*, volume 30, 2017.

[76] Francesco Visin, Marco Ciccone, Adriana Romero, Kyle Kastner, Kyunghyun Cho, Yoshua Bengio, Matteo Matteucci, and Aaron Courville. Reseg: A recurrent neural network-based model for semantic segmentation. In *CVPRW*, pages 41–48, 2016.

[77] Francesco Visin, Kyle Kastner, Kyunghyun Cho, Matteo Matteucci, Aaron Courville, and Yoshua Bengio. Renet: A recurrent neural network based alternative to convolutional networks. *arXiv preprint arXiv:1505.00393*, 2015.

[78] Haohan Wang, Songwei Ge, Zachary Lipton, and Eric P Xing. Learning robust global representations by penalizing local predictive power. In *NeurIPS*, volume 32, 2019.

[79] 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 *ICCV*, 2021.

[80] Ross Wightman. Pytorch image models. `https://github.com/rwightman/pytorch-image-models`, 2019.

[81] Tan Yu, Xu Li, Yunfeng Cai, Mingming Sun, and Ping Li. S^2-mlpv2: Improved spatial-shift mlp architecture for vision. *arXiv preprint arXiv:2108.01072*, 2021.

[82] Tan Yu, Xu Li, Yunfeng Cai, Mingming Sun, and Ping Li. S^2-mlp: Spatial-shift mlp architecture for vision. In *WACV*, pages 297–306, 2022.

[83] Weihao Yu, Mi Luo, Pan Zhou, Chenyang Si, Yichen Zhou, Xinchao Wang, Jiashi Feng, and Shuicheng Yan. Metaformer is actually what you need for vision. In *CVPR*, 2022.

[84] Li Yuan, Yunpeng Chen, Tao Wang, Weihao Yu, Yujun Shi, Zi-Hang Jiang, Francis EH Tay, Jiashi Feng, and Shuicheng Yan. Tokens-to-token vit: Training vision transformers from scratch on imagenet. In *ICCV*, pages 558–567, 2021.

[85] Sangdoo Yun, Dongyoon Han, Seong Joon Oh, Sanghyuk Chun, Junsuk Choe, and Youngjoon Yoo. Cutmix: Regularization strategy to train strong classifiers with localizable features. In *ICCV*, pages 6023–6032, 2019.

[86] David Junhao Zhang, Kunchang Li, Yunpeng Chen, Yali Wang, Shashwat Chandra, Yu Qiao, Luoqi Liu, and Mike Zheng Shou. Morphmlp: A self-attention free, mlp-like backbone for image and video. *arXiv preprint arXiv:2111.12527*, 2021.

[87] Hongyi Zhang, Moustapha Cisse, Yann N Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. In *ICLR*, 2018.

[88] Zhun Zhong, Liang Zheng, Guoliang Kang, Shaozi Li, and Yi Yang. Random erasing data augmentation. In *AAAI*, volume 34, pages 13001–13008, 2020.

[89] Bolei Zhou, Hang Zhao, Xavier Puig, Sanja Fidler, Adela Barriuso, and Antonio Torralba. Scene parsing through ade20k dataset. In *CVPR*, pages 633–641, 2017.

[90] Daquan Zhou, Bingyi Kang, Xiaojie Jin, Linjie Yang, Xiaochen Lian, Zihang Jiang, Qibin Hou, and Jiashi Feng. Deepvit: Towards deeper vision transformer. *arXiv preprint arXiv:2103.11886*, 2021.

[91] Daquan Zhou, Yujun Shi, Bingyi Kang, Weihao Yu, Zihang Jiang, Yuan Li, Xiaojie Jin, Qibin Hou, and Jiashi Feng. Refiner: Refining self-attention for vision transformers. *arXiv preprint arXiv:2106.03714*, 2021.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] The throughputs for our models are a bit inferior because of using RNNs, see Section 4.1 and 5.
   (c) Did you discuss any potential negative societal impacts of your work? [Yes] See Appendix A.

(d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]

2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] Included in supplemental material.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes]
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [No] We follow the typical practice used in our baseline methodology no error bars are reported.
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [N/A] They are included only if they are of essential significance to each experiment/item.
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes]
   (b) Did you mention the license of the assets? [N/A]
   (c) Did you include any new assets either in the supplemental material or as a URL? [No]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

A Societal impact

The impact of this study on society has both positive and negative aspects. Here we discuss each.

On the positive side, our proposal would promote modeling methods using LSTMs in computer vision. This study takes image patches as tokens and models their relationships with LSTMs. Although LSTMs have been used in computer vision, designing image recognition with a module that includes LSTMs in the spatial direction as the main elements, as our study does, is new. It is exciting to see if this design benefits computer vision tasks other than image classification. Thus, our study would be an impetus for further research on its application to various computer vision tasks.

On the other side, our architecture may increase the carbon dioxide footprint: the study of new architectures for vision, such as Sequencer, requires iterative training of models for long periods to optimize the model's design. In particular, Sequencer is not a FLOPs-friendly design, and the amount of carbon dioxide emitted during training is likely to be high. Therefore, considering the environmental burden caused by the training of Sequencers, research to reduce the computational cost of Sequencers is also desired by society.

B Implementation details

In this section, implementation details are supplemented. We describe the pseudocode of the BiLSTM2D layer, the architecture details, settings for training on IN-1K, and introduce settings for transfer learning.

B.1 Pseudocode

Algorithm 1 Pseudocode of BiLSTM2D layer.

```
# B: batch size H: height, W: width, C: channel, D: hidden dimension
# x: input tensor of shape (B, H, W, C)
### initialization ###
self.rnn_v = nn.LSTM(C, D, num_layers=1, batch_first=True, bias=True, bidirectional=True)
self.rnn_h = nn.LSTM(C, D, num_layers=1, batch_first=True, bias=True, bidirectional=True)
self.fc = nn.Linear(4 * D, C)
### forward ###
def forward(self, x):
    v, _ = self.rnn_v(x.permute(0, 2, 1, 3).reshape(-1, H, C))
    v = v.reshape(B, W, H, -1).permute(0, 2, 1, 3)
    h, _ = self.rnn_h(x.reshape(-1, W, C))
    h = h.reshape(B, H, W, -1)
    x = torch.cat([v, h], dim=-1)
    x = self.fc(x)
    return x
```

B.2 Architecture details

This subsection describes Sequencer's architecture. The architectural details are shown in Table 4 and 5.

Sequencer2D-S is based on a ViP-S/7-like architecture. We intend to directly compare the BiLSTM2D layer in Sequencer2D, which has a similar structure, with the Permute-MLP layer in ViP-S/7. Table 4 is a summary of the architecture. In keeping with ViP, the first stage of Sequencers involves patch embedding with a 7x7 kernel. The second stage of Sequencers performs patch embedding with a 2x2 kernel, but the following two stages have no downsampling. The classifier of Sequencers then continues with layer normalization (LN) [1], followed by global average pooling and a linear layer. The number of blocks of Sequencer2D-S, Sequencer2D-M, and Sequencer2D-L correspond to ViP-S/7, ViP-M/7, and ViP-L/7, respectively. However, as described in Appendix C, we configure the dimension of the block to be different from ViP-M/7 and ViP-L/7 for Sequencer2D-M and Sequencer2D-L, respectively, because high dimension causes over-fitting.

VSequencer is a bit different from Sequencer2D in that it is non-hierarchical architecture. Table 5 define that no downsampling is performed in the second stage, instead of downsampling with 14x14 kernel for patch embedding in the first stage. In addition, we match the dimension of the blocks in the first stage to the dimension of the subsequent blocks.

Following the overall architecture, we describe the details of the modules not mentioned in the main text. Sequencer2D block and the Vanilla Sequencer block use LNs [1] for the normalization layers. We follow previous studies for the channel MLPs of these blocks and employ MLPs with Gaussian Error Linear Units (GELUs) [25] as the activation function; the ratio of increasing dimension in MLPs is uniformly 3x, as shown in Table 4 and 5.

B.3 IN-1K settings

On IN-1K dataset [39], we utilize the hyper-parameters displayed in Table 6 to scratch train models in subsections 4.1 and 4.3. All Sequencer variants, including the models in the ablation study, follow almost the same settings for pre-training. However, the stochastic depth rate and batch size are adjusted depending on the model variant. The models in the ablation study are Sequencer2D-S based because of the following Sequencer2D-S settings.

The fine-tuning Sequencer2D-L$\uparrow 392^2$ in subsection 4.2 has slightly different hyper-parameters than the pre-training models. There are changes in the settings for the number of epochs and learning rate because it uses trained weights, so there is no need to increase these hyper-parameters. In addition, we used crop ratio 0.875 during testing in the pre-training models instead of crop ratio 1.0 in the fine-tuning model.

B.4 Transfer learning settings

Details of the datasets used for transfer learning in subsection 4.4 are shown in Table 7. This summary includes for each dataset CIFAR-10 [38], CIFAR-100 [38], Flowers-102 [55], and Stanford Cars [37], the number of training images, test images, and number of categories are listed.

Table 6 demonstrates the hyperparameters used in transfer learning with these datasets. The training epochs are especially adjusting to the datasets and changing them. The reason for this is attributable to the different sizes of the datasets.

C More results

This section discusses additional results that could not be addressed in the main text. The contents of the experiment consist of three parts: an evaluation of robustness in subsection C.1, an evaluation of generalization performance in subsection C.2, and a discussion of over-fitting in subsection C.3.

C.1 Robustness

In this subsection, we evaluate the robustness of Sequencer. There are two main evaluation methods, benchmark datasets and adversarial attacks.

Evaluation with benchmark datasets reveals nice robustness of Sequencer. The evaluation results are summarized in Table 8. We test our models, trained on only IN-1K, on several datasets such as ImageNet-A/R/Sketch/C (IN-A/R/Sketch/C) [26, 23, 78, 24] to evaluate robustness. We evaluate our models on IN-C with mean corruption error (mCE), and on other datasets with top-1 accuracy. This result leads us to suggest that for models with a similar number of parameters, Sequencer is conquered by Swin and is robust enough to be competitive with ConvNeXt. Table 9 shows detail evaluation on IN-C. According to the results, it is understood that Sequencer is more immune to corruptions other than Noise than Swin and ConvNeXt, and, in particular, the model is less sensitive to weather conditions.

Sequencers are tolerant of principal adversarial attacks. We evaluate robustness using the single-step attack algorithm FGSM [18] and multi-step attack algorithm PGD [52]. Both algorithms give a perturbation of max magnitude 1. For PGD, we choose steps 5 and step size 0.5. This setup is based on RVT [53]. Table 9 indicates that Sequencer2D-L defeats in both FGSM and PGD compared to other models. Thus, Sequencer has an advantage over conventional models, such as RVT, which tout robustness on these adversarial attacks.

Table 4: **Variants of Sequencer2D and details**. "d" denotes the input/output dimension, and D denotes the hidden dimension as above. "$\downarrow n$" (e.g., $\downarrow 2$) shows the stride of the downsampling is n

	Sequencer2D-S	Sequencer2D-M	Sequencer2D-L
stage 1	Patch Embedding$\downarrow 7$ [BiLSTM2D: 192d $D = 48$ MLP: 3 exp. ratio] $\times 4$	Patch Embedding$\downarrow 7$ [BiLSTM2D: 192d $D = 48$ MLP: 3 exp. ratio] $\times 4$	Patch Embedding$\downarrow 7$ [BiLSTM2D: 192d $D = 48$ MLP: 3 exp. ratio] $\times 8$
stage 2	Patch Embedding$\downarrow 2$ [BiLSTM2D: 384d $D = 96$ MLP: 3 exp. ratio] $\times 3$	Patch Embedding$\downarrow 2$ [BiLSTM2D: 384d $D = 96$ MLP: 3 exp. ratio] $\times 3$	Patch Embedding$\downarrow 2$ [BiLSTM2D: 384d $D = 96$ MLP: 3 exp. ratio] $\times 8$
stage 3	Point-wise Linear [BiLSTM2D: 384d $D = 96$ MLP: 3 exp. ratio] $\times 8$	Point-wise Linear [BiLSTM2D: 384d $D = 96$ MLP: 3 exp. ratio] $\times 14$	Point-wise Linear [BiLSTM2D: 384d $D = 96$ MLP: 3 exp. ratio] $\times 16$
stage 4	Point-wise Linear [BiLSTM2D: 384d $D = 96$ MLP: 3 exp. ratio] $\times 3$	Point-wise Linear [BiLSTM2D: 384d $D = 96$ MLP: 3 exp. ratio] $\times 3$	Point-wise Linear [BiLSTM2D: 384d $D = 96$ MLP: 3 exp. ratio] $\times 4$
classifier	Layer Norm., Global Average Pooling, Linear		

C.2 Generalization ability

The generalization ability of Sequencers is also impressive. We evaluate our models on ImageNet-Real/V2 (IN-Real/V2) [2, 62] to test their generalization performance: IN-Real is a re-labeled dataset of the IN-1K validation set, and IN-V2 is the dataset that re-collects the IN-1K validation set. Table 8 shows the results of evaluating the top-1 accuracy on both datasets. We reveal an understanding of the Sequencer's excellent generalization ability.

C.3 Over-fitting

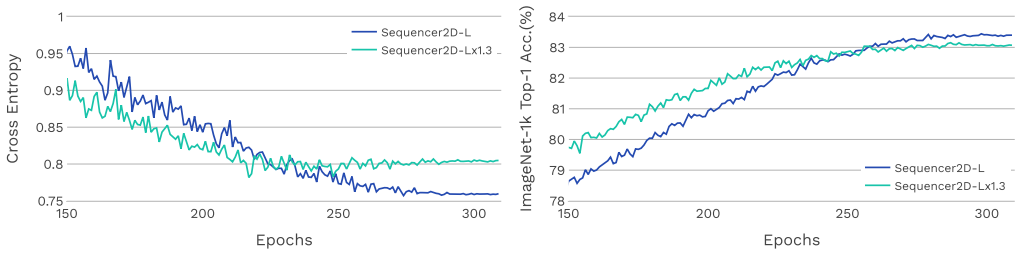
Wide Sequencers tend to be over-trained. We scratch-train Sequencer2D-Lx1.3, which has 4/3 times the dimension of each layer of Sequencer2D-L, on IN-1K. The training utilizes the same conditions as Sequencer2D-L. Consequently, as Table 10 shows, Sequencer2D-Lx1.3 has 0.8% less accuracy than Sequencer2D-L. Figure 6 illustrates the cross-entropy evolution and top-1 accuracy on IN-1K validation set for the two models. On the one hand, cross-entropy decreased on Sequencer2D-L in the last 100 epochs. On the other hand, Sequencer2D-Lx1.3 is increasing. Thus, widening Sequencer is counterproductive for training.

C.4 Semantic segmentation

We evaluate models with Sequencer as the backbone for a semantic segmentation task. We trained and evaluated on ADE20K dataset [89], a well-known scene parsing benchmark. The dataset consists of the training set with about 20k images and the validation set with about 2k, covering 150 fine-grained semantic classes. We employed Sequencer as the backbone of SemanticFPN [36] to train and evaluate semantic segmentation. The training adopts a batch size of 32 and AdamW [50] with the initial learning rate of 2e-4, decay in the polynomial decay schedule with a power of 0.9, and 40k iterations

Table 5: **Variants of VSequencer and details**. "d" denotes the input/output dimension, and D denotes the hidden dimension as above. "$\downarrow n$" (e.g., $\downarrow 2$) shows the stride of the downsampling is n

	VSequencer-S	VSequencer-S(H)	VSequencer-S(PE)
stage 1	Patch Embedding$\downarrow$14 [BiLSTM: 384d $D = 192$ MLP: 3 exp. ratio] $\times 4$	Patch Embedding$\downarrow$7 [BiLSTM: 192d $D = 96$ MLP: 3 exp. ratio] $\times 4$	Patch Embedding$\downarrow$14 [BiLSTM: 384d $D = 192$ MLP: 3 exp. ratio] $\times 4$
stage 2	Point-wise Linear [BiLSTM: 384d $D = 192$ MLP: 3 exp. ratio] $\times 3$	Patch Embedding$\downarrow$2 [BiLSTM: 384d $D = 192$ MLP: 3 exp. ratio] $\times 3$	Point-wise Linear [BiLSTM: 384d $D = 192$ MLP: 3 exp. ratio] $\times 3$
stage 3	Point-wise Linear [BiLSTM: 384d $D = 192$ MLP: 3 exp. ratio] $\times 8$	Point-wise Linear [BiLSTM: 384d $D = 192$ MLP: 3 exp. ratio] $\times 8$	Point-wise Linear [BiLSTM: 384d $D = 192$ MLP: 3 exp. ratio] $\times 8$
stage 4	Point-wise Linear [BiLSTM: 384d $D = 192$ MLP: 3 exp. ratio] $\times 3$	Point-wise Linear [BiLSTM: 384d $D = 192$ MLP: 3 exp. ratio] $\times 3$	Point-wise Linear [BiLSTM: 384d $D = 192$ MLP: 3 exp. ratio] $\times 3$
classifier	Layer Norm., Global Average Pooling, Linear		



(a) Cross entropy

(b) Top-1 accuracy

Figure 6: **Comparison of different model widths**. (a) is cross entropy, (b) is top-1 accuracy comparison, on IN-1K validation set. **The blue curve** represents the original Sequencer2D-L, which did not produce any problems and is learning all the way through. In contrast, **the green curve** represents the wider Sequencer2D-Lx1.3. This model stalls in the second half and is somewhat degenerate.

of training. These settings follow Metaformer [83]. Table 3 of the result indicates that Sequencer has the generalization for segmentation is comparable to other leading models.

C.5 Object Detection

We evaluate Sequencer on COCO benchmark [45]. The dataset consists of 118k training images and 5k validation images. Sequencer with ImageNet pre-trained weights is employed as the backbone of RetinaNet [44]. Following [44], we employ AdamW, batch size of 16, and $1\times$ training schedule. Table 11 shows that Sequencer is not suited for existing standard object detection models such as

Table 6: **Hyper-parameters**. ↑ denotes fine-tuning pre-trained model on IN-1K. Multiple values are for each model, respectively.

Training config.	Sequencer2D-S/M/L 224^2	Sequencer2D-L↑ 392^2	Sequencer2D-S↑/M↑/L↑ 224^2
dataset	IN-1K [39]	IN-1K [39]	CIFAR$^{10,\ 100}$, Flowers, Cars
optimizer	AdamW [50]	AdamW [50]	AdamW [50]
base learning rate	2e-3/1.5e-3/1e-3	5e-5	1e-4
weight decay	0.05	1e-8	1e-4
optimizer ϵ	1e-8	1e-8	1e-8
optimizer momentum	$\beta_1, = 0.9, \beta_2$=0.999	$\beta_1, = 0.9, \beta_2$=0.999	$\beta_1, = 0.9, \beta_2$=0.999
batch size	2048/1536/1024	512	512
training epochs	300	30	CIFAR: 200, Others: 1000
learning rate schedule	cosine decay	cosine decay	cosine decay
lower learning rate bound	1e-6	1e-6	1e-6
warmup epochs	20	None	5
warmup schedule	linear	None	linear
warmup learning rate	1e-6	None	1e-6
cooldown epochs	10	None	10
crop ratio	0.875	1.0	0.875
randaugment [11]	(9, 0.5)	(9, 0.5)	(9, 0.5)
mixup α [87]	0.8	0.8	0.8
cutmix α [85]	1.0	1.0	1.0
random erasing [88]	0.25	0.25	None
label smoothing [66]	0.1	0.1	0.1
stochastic depth [30]	0.1/0.2/0.4	0.4	0.1/0.2/0.4
gradient clip	None	None	1

Table 7: **Transfer learning datasets**.

Dataset	Train Size	Test size	#Classes
CIFAR-10 [38]	50,000	10,000	10
CIFAR-100 [38]	50,000	10,000	100
Flowers-102 [55]	2,040	6,149	102
Stanford Cars [37]	8,144	8,041	196

RetinaNet. It shows no improvement trend for model scaling. It also struggles to detect small objects, making RNN-based object detection models an issue to consider in the future.

C.6 More studies

Method of merge As shown in Figure 2, "`concatenate`" is used to merge the vertical BiLSTM and horizontal BiLSTM outputs but "`add`" can also be used. See Table 12a for the result of the experiment.

D Effective receptive field

This section covers in detail the effective receptive fields (ERFs) [51] used in the visualization in subsection 4.5. First, we explain how the visualized effective receptive fields are obtained. Second, we present other visualization results not addressed in the main text. The ERF's calculations in this paper are based on [14].

D.1 Calculation of visualized ERFs

The ERF [51] is a technique for calculating the pixels that contribute to the center of a output feature maps of a neural network. Let $\mathbf{I} \in \mathbb{R}^{n \times h \times w \times c}$ be a input image collection and $\mathbf{O} \in \mathbb{R}^{n \times h' \times w' \times c'}$ be the output feature map collection. The center of the output feature map can be expressed as $\mathbf{O}_{:, \lfloor h'/2 \rfloor, \lfloor w'/2 \rfloor, :}$, where $\lfloor \cdot \rfloor$ is the floor function. Each element of the derivative of $\mathbf{O}_{i, \lfloor h'/2 \rfloor, \lfloor w'/2 \rfloor, j}$ to $\mathbf{I}$, i.e., $\frac{\partial \left(\sum_{i,j} \mathbf{O}_{i, \lfloor h'/2, \rfloor \lfloor w'/2 \rfloor, j} \right)}{\partial \mathbf{I}}$, represents to what extent the center of the output feature map

Table 8: **The robustness** is evaluated on IN-A [26] (top-1 accuracy), IN-R [23] (top-1 accuracy), IN-Sketch [78] (top-1 accuracy), IN-C [24] (mCE), FGSM [18] (top-1 accuracy), and PGD [52] (top-1 accuracy). **The generalization ability** is evaluated on IN-Real [2] and IN-V2 [62]. We denote the higher as better value as ↑ and the lower as better value as ↓. Rather than those reported in the original paper, the values we observed are marked with †. If the model name has †, it means that we observed all the metrics of the model.

Model	#Param.	FLOPs	Clean(↑)	A(↑)	R(↑)	Sk.(↑)	C(↓)	FGSM(↑)	PGD(↑)	Real(↑)	V2(↑)
Swin-T [48]	28M	4.5G	81.2	21.6	41.3	29.1	62.0	33.7	7.3	86.7†	69.6†
ConvNeXt-T [49]	29M	4.5G	82.1	24.2	47.2	33.8	53.2	37.8†	10.5†	87.3†	71.0†
RVT-S* [53]	23M	4.7G	81.9	25.7	47.7	34.7	49.4	51.8	28.2	-	-
Sequencer2D-S	28M	8.4G	82.3	26.7	45.1	33.4	53.0	49.2	25.0	87.4	71.8
Sequencer2D-M	38M	11.1G	82.8	30.5	46.3	34.7	51.8	50.8	26.3	87.6	72.5
Swin-S [48]†	50M	8.7G	83.2	32.5	45.2	32.3	54.9	45.9	18.1	87.7	72.1
ConvNeXt-S† [49]	50M	8.7G	83.1	31.3	**49.6**	**37.1**	49.5	46.1	17.7	**88.1**	72.5
Sequencer2D-L	54M	16.6G	**83.4**	**35.5**	48.1	35.8	**48.9**	**53.1**	**30.9**	87.9	**73.4**
Swin-B [48]	88M	15.4G	83.4	35.8	46.6	32.4	54.4	49.2	21.3	89.2†	75.6†
ConvNeXt-B [49]	89M	15.4G	83.8	36.7	51.3	38.2	46.8	47.5†	18.3†	88.4†	73.7†
RVT-B* [53]	92M	17.7G	82.6	28.5	48.7	36.0	46.8	53.0	29.9	-	-

Table 9: Details of robustness evaluation with IN-C.

		Noise			Blur				Weather				Digital			
Model	mCE	Gauss.	Shot	Impulse	Defocus	Glass	Motion	Zoom	Snow	Frost	Fog	Bright	Contrast	Elastic	Pixel	JPEG
Swin-S [48]	54.9	42.9	44.9	43.2	61.3	74.1	56.6	67.5	50.8	48.5	46.0	44.1	42.1	68.9	62.1	70.7
ConvNeXt-S [49]	49.5	**38.1**	**39.1**	**37.9**	57.8	72.5	**51.8**	**61.9**	46.1	43.8	44.6	39.6	37.6	66.7	55.1	**50.1**
Sequencer2D-L	**48.9**	43.3	42.0	41.4	**55.2**	**71.0**	**51.8**	63.3	**44.2**	**41.0**	**41.9**	**37.1**	**33.8**	**66.6**	**50.4**	51.1

changes for each perturbation of each pixel in each input image. Adding these together for all images and channels, we can calculate the average pixel contribution for all input images, which can be activated with a Rectified Linear Unit (ReLU) to get the positively contributing pixel values $\mathbf{P} \in \mathbb{R}^{n \times h \times w \times c}$, defined by

$$\mathbf{P} = \mathrm{ReLU}\left(\frac{\partial \left(\sum_{i,j} \mathbf{O}_{i,\lfloor h'/2, \rfloor \lfloor w'/2 \rfloor, j} \right)}{\partial \mathbf{I}} \right) . \tag{8}$$

Furthermore, the score $\mathrm{S} \in \mathbb{R}^{h \times w}$ is calculated by

$$\mathrm{S} = \log_{10} \left(\sum_{i,j} \mathbf{P}_{i,:,:,j} + 1 \right) , \tag{9}$$

and S is called the effective receptive field.

Next, define a visualized effective receptive field based on the effective receptive field. We want to compare the effective receptive fields across models. We, therefore, calculate the score $\mathrm{S}_{\mathrm{model}}$ for each model and rescale $\mathrm{S}_{\mathrm{model}}$ from 0 to 1 across the models. The tensor calculated in this way is called the visualized effective receptive field.

The derivatives used in these definitions are efficient if they take advantage of the auto-grad mechanism. Indeed, we also relied on the automatic auto-grad function on `PyTorch` [56] to calculate the effective receptive fields.

Table 10: **Comparison of accuracy for different model widths**.

Model	#Params.	FLOPs	Acc.
Sequencer2D-L	54M	16.6G	83.4
Sequencer2D-Lx1.3	96M	29.4G	83.0

Table 11: Object detection results on COCO dataset [45]

Backbone	Params (M)	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
ResNet-18 [22]	21.3	31.8	49.6	33.6	16.3	34.3	43.2
PoolFormer-S12 [83]	21.7	36.2	56.2	38.2	20.8	39.1	48.0
Sequencer2D-S	37.3	33.6	54.8	34.8	15.3	37.5	50.2
ResNet-50 [22]	37.7	36.3	55.3	38.6	19.3	40.0	48.8
PoolFormer-S24 [83]	31.1	38.9	59.7	41.3	23.3	42.1	51.8
Sequencer2D-M	47.9	34.5	55.5	35.9	15.0	39.0	51.6
ResNet-101 [22]	56.7	38.5	57.8	41.2	21.4	42.6	51.1
PoolFormer-S36 [83]	40.6	39.5	60.5	41.8	22.5	42.9	52.4
Sequencer2D-L	63.9	35.0	56.4	36.5	16.5	39.6	51.6

Table 12: **More Sequencer ablation experiments**.

(a) Method of merge

Union	#Params.	FLOPs	Acc.
`add`	27M	8.0G	82.2
`concatnate`	28M	8.4G	**82.3**

D.2 More visualization of ERFs

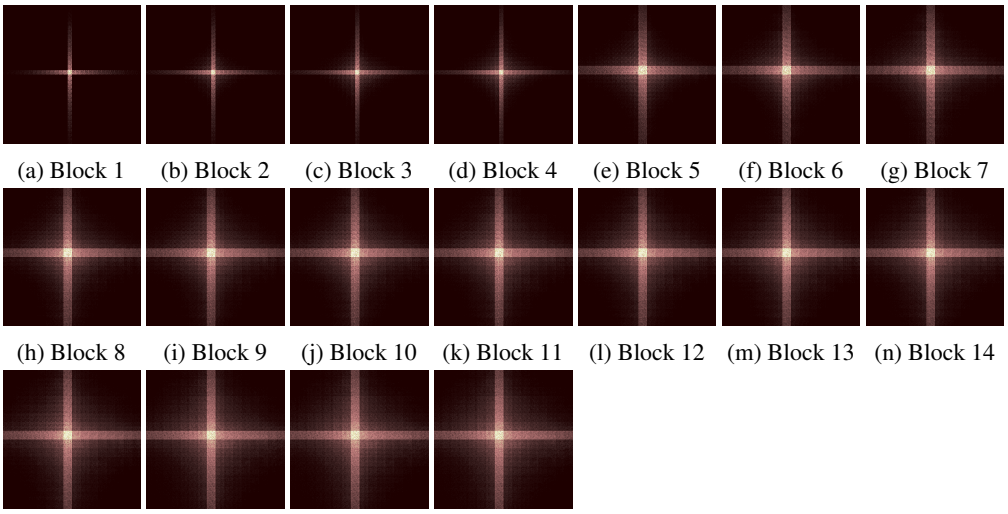
We introduce additional visualization and concrete visualization method. We experiment with visualization using input images of two different resolutions.

We visualize the effective receptive fields of Sequencer2D-S and comparative models by using 224^2 resolution images. The method is applied to the following models for comparing: ResNet-50 [22], ConvNeXt-T [49], CycleMLP-B2 [7], DeiT-S [72], Swin-T [48], GFNet-S [61], and ViP-S/7 [28]. The object to be visualized is the output for each block, and the effective receptive fields are calculated. For example, in the case of Sequencer2D-S, the effective receptive fields are calculated for the output of each Sequencer block. We are rescaling within a value between 0 and 1 for the whole to effective receptive fields for each model block.

The effective receptive fields of Sequencer2D-S and comparative models are then visualized using input images with a resolution of 448^2. The reason for running experiments is to verify how the receptive field is affected when the input resolution is increased compared to the 224^2 resolution input image. Sequencer2D-S compare with ResNet-50 [22], ConvNeXt-T [49], CycleMLP-B2 [7], DeiT-S [72], and GFNet-S [28]. The method of visualization of the effective receptive field follows the case of input images with a resolution of 224^2.

Sequencer has very distinctive cruciform ERFs in all layers. Table 7, 8, 9, 10, 11, 12, 13, and 14 illustrates this fact for 224^2 resolution input images. Furthermore, as shown in Table 15, 16, 17, 18 and 19, we observe the same trend when the double resolution. The ERFs are structurally quite different from the ERFs other than ViP, which have a similar structure. ViP's ERFs have, on average, some also coverage except for the cruciforms. In contrast, Sequencer's ERFs are limited to the cruciform and its neighborhood.

It is interesting to note that Sequencer, with its characteristic ERFs, achieves high accuracy. It will be helpful for future architecture development because of the possibility of creating Sequencer-like ERFs outside of LSTM.

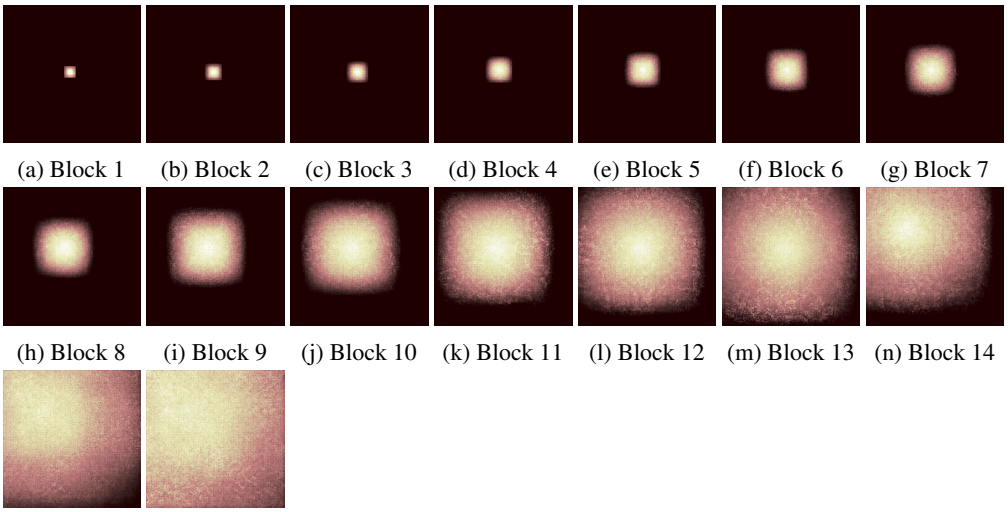


(a) Block 1 (b) Block 2 (c) Block 3 (d) Block 4 (e) Block 5 (f) Block 6 (g) Block 7

(h) Block 8 (i) Block 9 (j) Block 10 (k) Block 11 (l) Block 12 (m) Block 13 (n) Block 14

(o) Block 15 (p) Block 16 (q) Block 17 (r) Block 18

Figure 7: ERFs in Sequencer2D-S on images with resolution 224^2.



(a) Block 1 (b) Block 2 (c) Block 3 (d) Block 4 (e) Block 5 (f) Block 6 (g) Block 7

(h) Block 8 (i) Block 9 (j) Block 10 (k) Block 11 (l) Block 12 (m) Block 13 (n) Block 14

(o) Block 15 (p) Block 16

Figure 8: ERFs in ResNet-50 [22] on images with resolution 224^2.

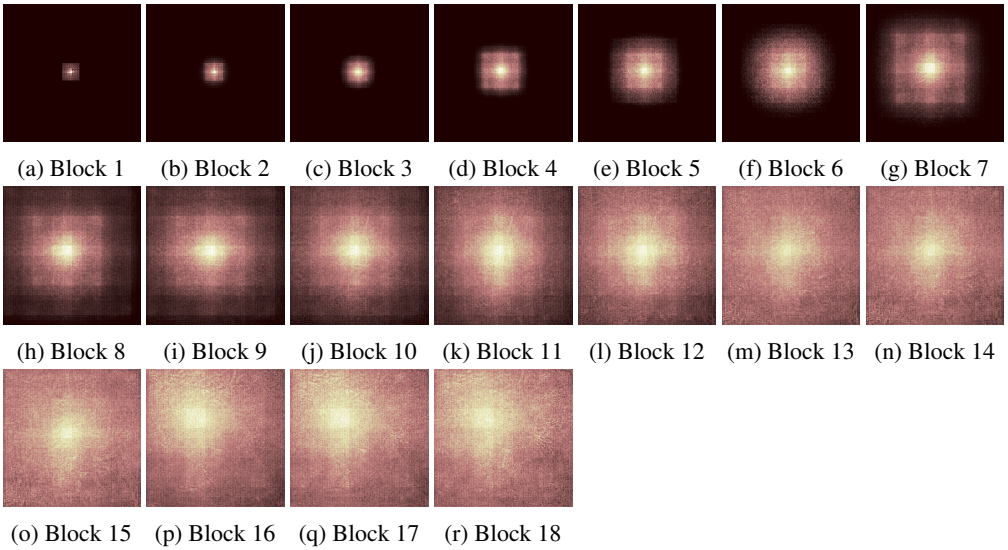


Figure 9: ERFs in ConvNeXt-T [49] on images with resolution 224^2.

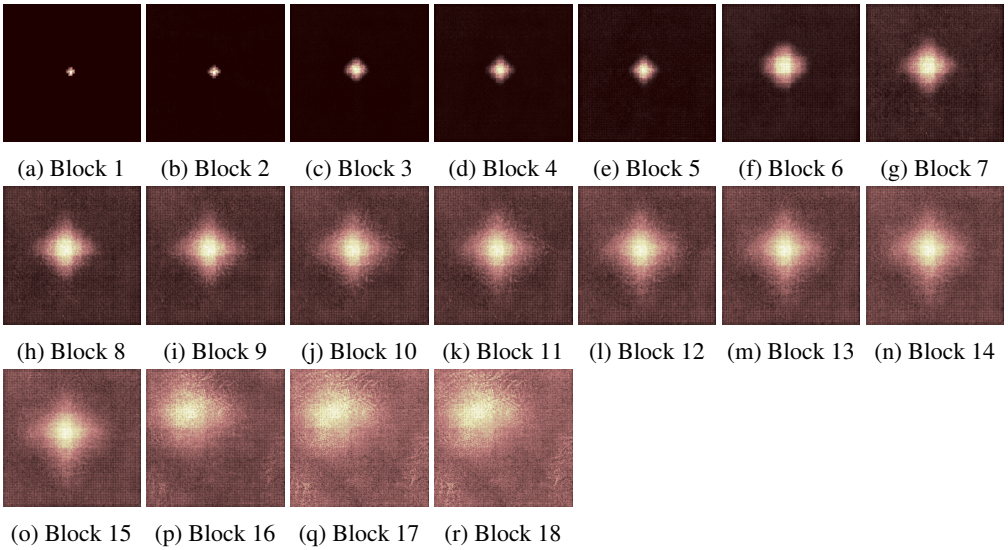


Figure 10: ERFs in CycleMLP-B2 [7] on images with resolution 224^2.

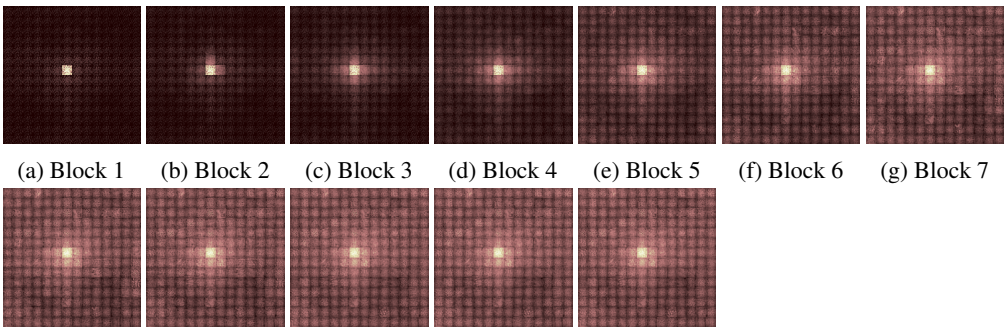
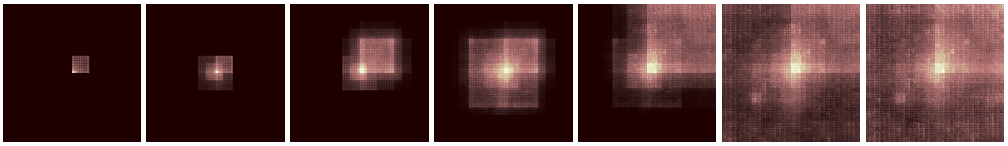
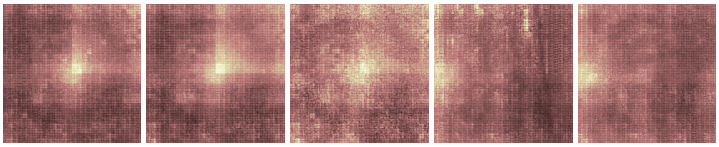


Figure 11: ERFs in DeiT-S [72] on images with resolution 224^2.

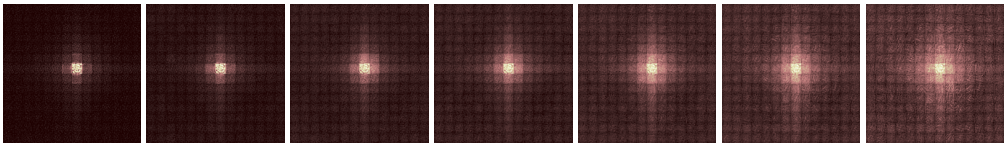


(a) Block 1 (b) Block 2 (c) Block 3 (d) Block 4 (e) Block 5 (f) Block 6 (g) Block 7

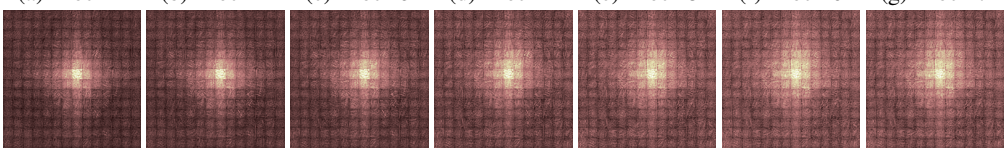


(h) Block 8 (i) Block 9 (j) Block 10 (k) Block 11 (l) Block 12

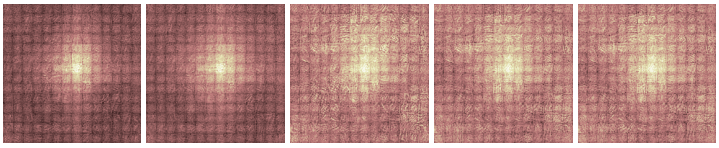
Figure 12: ERFs in Swin-T [48] on images with resolution 224^2.



(a) Block 1 (b) Block 2 (c) Block 3 (d) Block 4 (e) Block 5 (f) Block 6 (g) Block 7

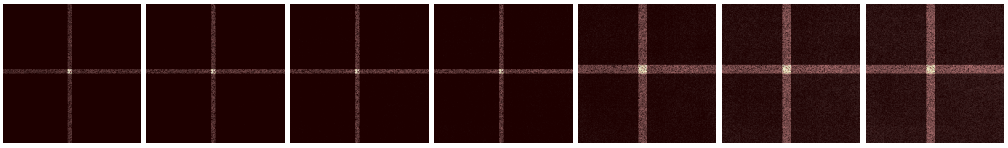


(h) Block 8 (i) Block 9 (j) Block 10 (k) Block 11 (l) Block 12 (m) Block 13 (n) Block 14

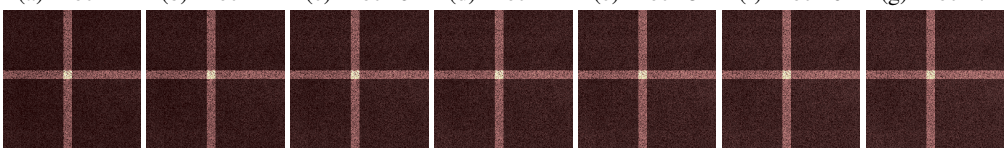


(o) Block 15 (p) Block 16 (q) Block 17 (r) Block 18 (s) Block 19

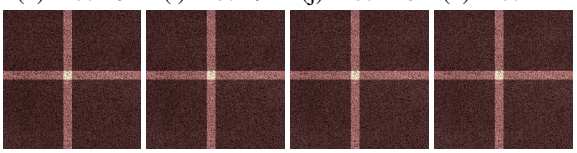
Figure 13: ERFs in GFNet-S [61] on images with resolution 224^2.



(a) Block 1 (b) Block 2 (c) Block 3 (d) Block 4 (e) Block 5 (f) Block 6 (g) Block 7

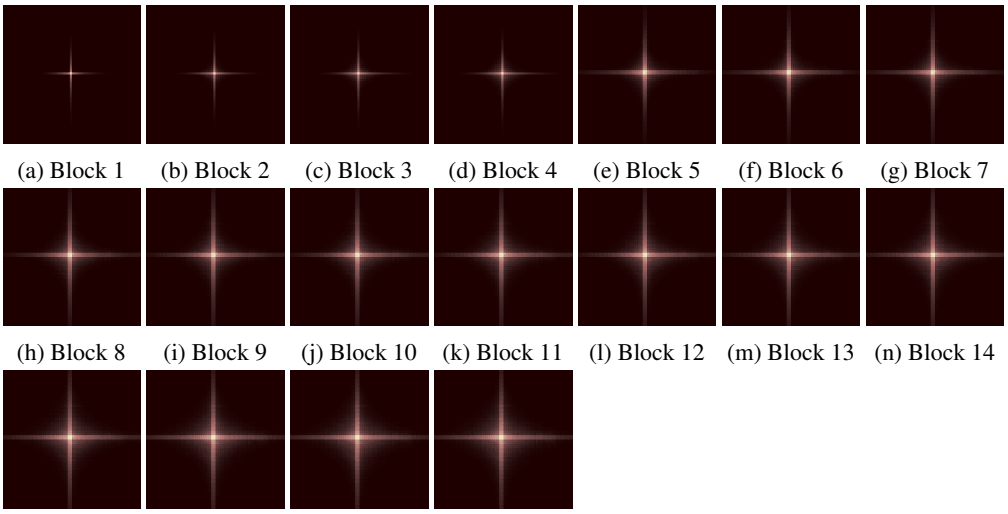


(h) Block 8 (i) Block 9 (j) Block 10 (k) Block 11 (l) Block 12 (m) Block 13 (n) Block 14



(o) Block 15 (p) Block 16 (q) Block 17 (r) Block 18

Figure 14: ERFs in ViP-S/7 [28] on images with resolution 224^2.

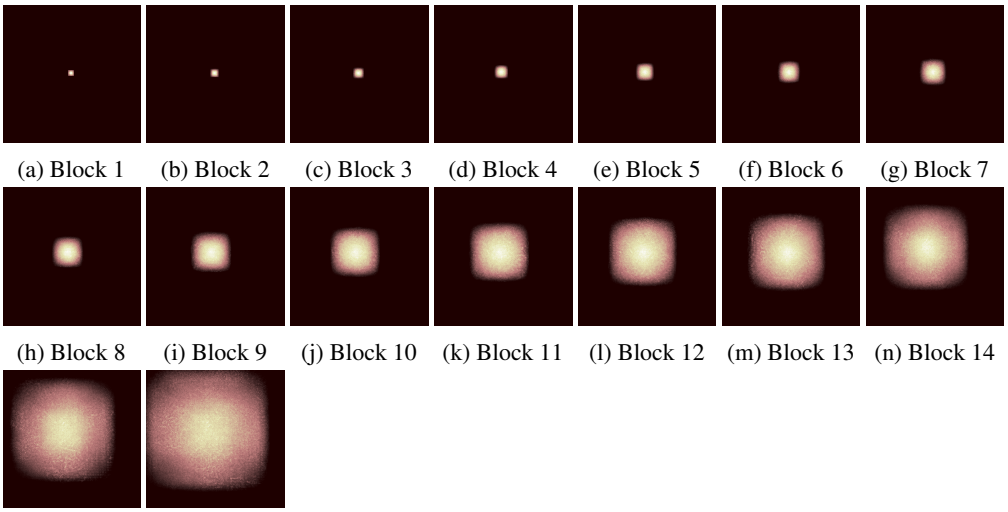


(a) Block 1 (b) Block 2 (c) Block 3 (d) Block 4 (e) Block 5 (f) Block 6 (g) Block 7

(h) Block 8 (i) Block 9 (j) Block 10 (k) Block 11 (l) Block 12 (m) Block 13 (n) Block 14

(o) Block 15 (p) Block 16 (q) Block 17 (r) Block 18

Figure 15: ERFs in Sequencer2D-S on images with resolution 448^2.



(a) Block 1 (b) Block 2 (c) Block 3 (d) Block 4 (e) Block 5 (f) Block 6 (g) Block 7

(h) Block 8 (i) Block 9 (j) Block 10 (k) Block 11 (l) Block 12 (m) Block 13 (n) Block 14

(o) Block 15 (p) Block 16

Figure 16: ERFs in ResNet-50 [22] on images with resolution 448^2.

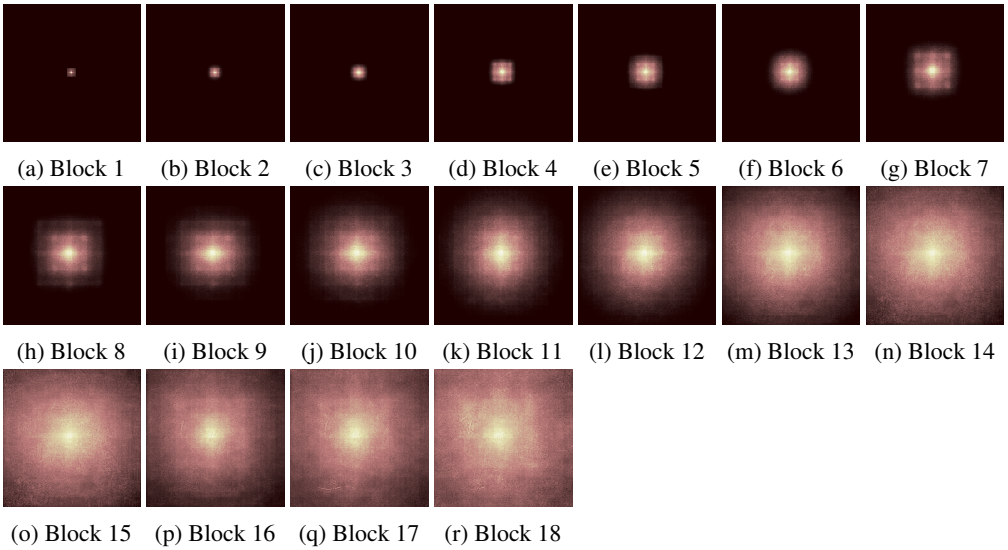


Figure 17: ERFs in ConvNeXt-T [49] on images with resolution 448^2.

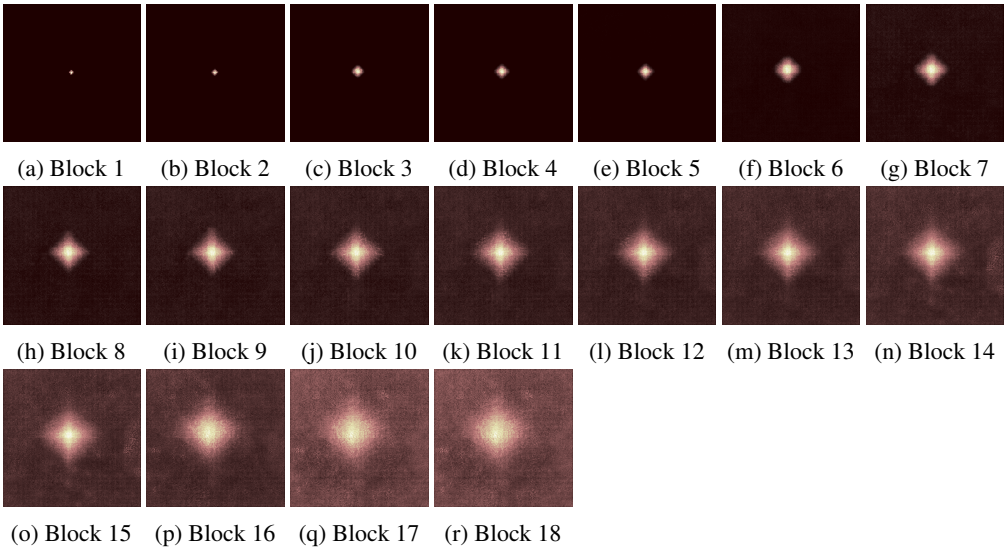


Figure 18: ERFs in CycleMLP-B2 [7] on images with resolution 448^2.

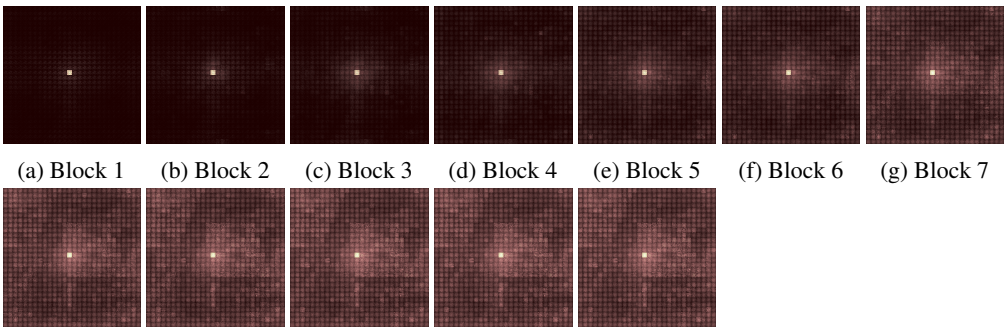


Figure 19: ERFs in DeiT-S [72] on images with resolution 448^2.