LambdaNetworks: Modeling long-range Interactions without Attention

Medvedev Dmitry

Moscow, 2021

Lambda Networks

Architecture	Params (M)	Train (ex/s)	Infer (ex/s)	ImageNet top-1
LambdaResNet-152	**51**	**1620**	**6100**	86.7
EfficientNet-B7	66	170 (9.5x)	980 (6.2x)	86.7
ViT-L/16	307	180 (9.0x)	640 (9.5x)	**87.1**

Table 5: **Comparison of models trained on extra data**. ViT-L/16 is pre-trained on JFT and fine-tuned on ImageNet at resolution 384x384, while EfficientNet and LambdaResNet are co-trained on ImageNet and JFT pseudo-labels. Training and inference throughput is shown for 8 TPUv3 cores.

lambda-networks

Implementation of LambdaNetworks, a new approach to image recognition that reaches SOTA with less compute

Python ☆ 1.4k 147

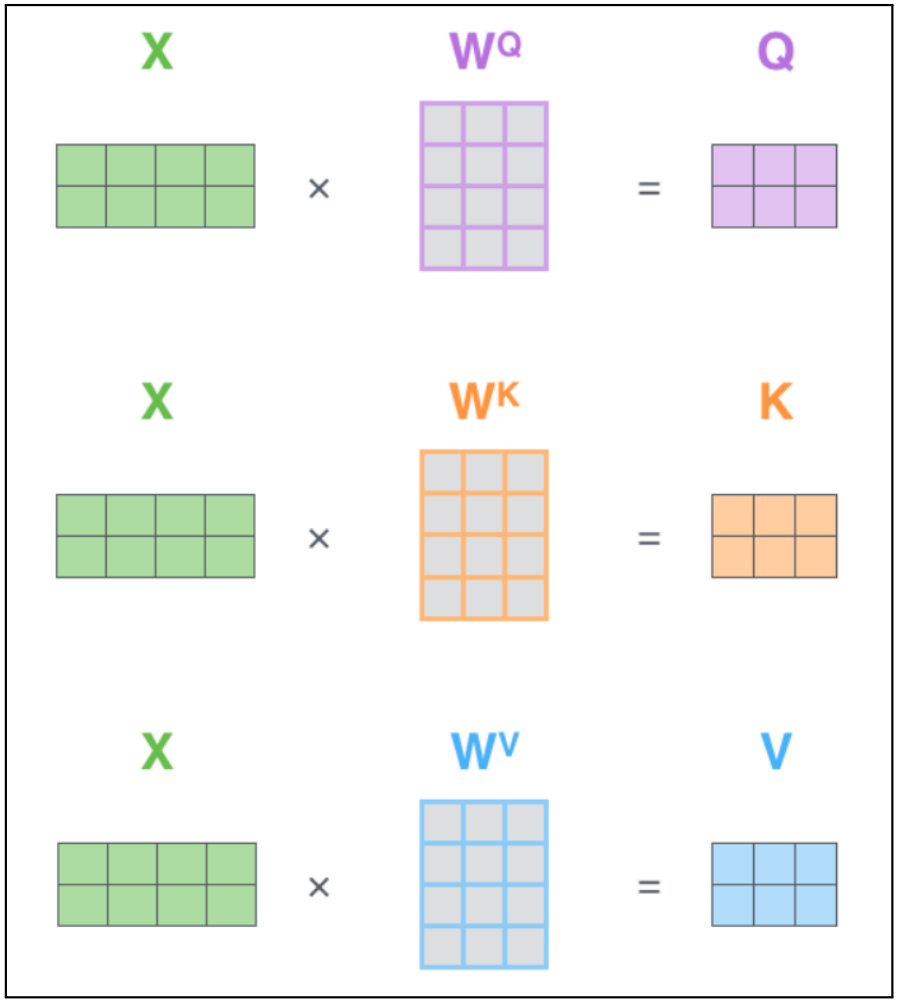
Final Decision

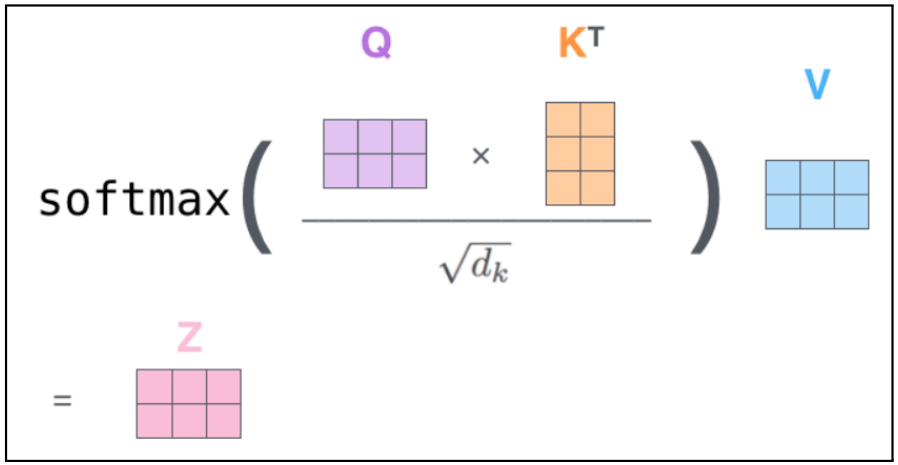
ICLR 2021 Conference Program Chairs

07 Jan 2021 (modified: 12 Jan 2021) ICLR 2021 Conference Paper476 Decision

Decision: Accept (Spotlight)

Self-Attention





General idea: linear attention + position embeddings

Aim: $\boxed{F((q_n, n), \mathbb{C}) \to y_n}$

$$Q \in \mathbb{R}^{|n| \times |k|}, \quad K \in \mathbb{R}^{|m| \times |k|}, \quad V \in \mathbb{R}^{|m| \times |v|}$$

$$\text{Attention}(Q, K, V) = \text{softmax}(QK^T)V \sim \phi(Q)(\phi(K^T)V) \tag{1}$$

$$y_n = \underbrace{(\phi(K)^T V)^T}_{\text{some function}} \phi(q_n) \tag{2}$$

$$\lambda_n = \sum_m (\overline{k_m} + e_{nm}) v_m^T = \underbrace{\text{softmax}(K, \text{dim} = m)^T V}_{\text{content lambda}} + \underbrace{E_n^T V}_{\text{position lambda}} \tag{3}$$

$$y_n = F((q_n, n), \mathbb{C}) = \lambda(\mathbb{C}, n)(q_n) = \lambda_n^T q_n = (\lambda^c + \lambda_n^p)^T q_n \tag{4}$$

Content vs position interactions

- A **content-based** interaction considers the content of the context but ignores the relation between the query position and the context (e.g. relative distance between two pixels).
- A **position-based** interaction considers the relation between the query position and the context position.

Content	Position	Params (M)	FLOPS (B)	top-1
✓	×	14.9	5.0	68.8
×	✓	14.9	11.9	78.1
✓	✓	14.9	12.0	78.4

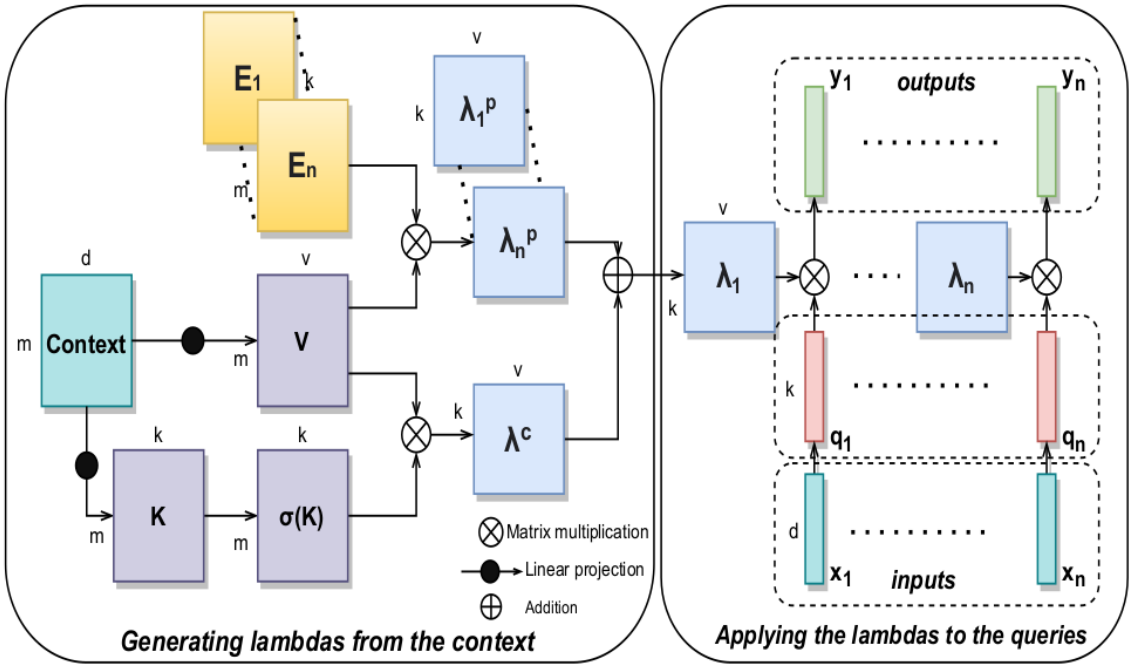
Table 10: **Contributions of content and positional interactions**. As expected, positional interactions are crucial to perform well on the image classification task.

Key Normalization

Normalization	top-1
Softmax on keys (default)	78.4
Softmax on keys & Softmax on queries	78.1
L2 normalization on keys	78.0
No normalization on keys	70.0
No batch normalization on queries and values	76.2

Table 12: **Impact of normalization schemes in the lambda layer.** Normalization of the keys along the context spatial dimension m, normalization of the queries along the query depth k.

Computational graph of the lambda layer

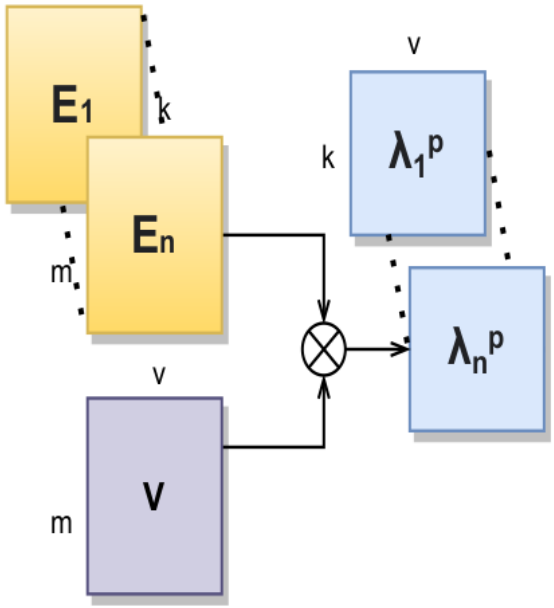


Name	Description
$\|k\|, \|v\|$	query, value depth
$\boldsymbol{X} \in \mathbb{R}^{\|n\| \times d}$	inputs
$\boldsymbol{C} \in \mathbb{R}^{\|m\| \times d}$	context
$\boldsymbol{Q} = \boldsymbol{X}\boldsymbol{W}_Q \in \mathbb{R}^{\|n\| \times \|k\|}$	queries
$\boldsymbol{K} = \boldsymbol{C}\boldsymbol{W}_K \in \mathbb{R}^{\|m\| \times \|k\|}$	keys
$\boldsymbol{V} = \boldsymbol{C}\boldsymbol{W}_V \in \mathbb{R}^{\|m\| \times \|v\|}$	values
$\sigma(\boldsymbol{K}) = \text{softmax}(\boldsymbol{K}, \text{axis}{=}m)$	normalized keys
$\boldsymbol{E}_n \in \mathbb{R}^{\|m\| \times \|k\|}$	relative position embeddings
$\boldsymbol{\lambda}^c = \bar{\boldsymbol{K}}^T \boldsymbol{V} \in \mathbb{R}^{\|k\| \times \|v\|}$	*content* lambda
$\boldsymbol{\lambda}_n^p = \boldsymbol{E}_n^T \boldsymbol{V} \in \mathbb{R}^{\|k\| \times \|v\|}$	*position* lambdas
$\boldsymbol{\lambda}_n = \boldsymbol{\lambda}^c + \boldsymbol{\lambda}_n^p \in \mathbb{R}^{\|k\| \times \|v\|}$	lambdas

Figure 2: **Computational graph of the lambda layer**. Contextual information for query position n is summarized into a lambda $\boldsymbol{\lambda}_n \in \mathbb{R}^{|k| \times |v|}$. Applying the lambda dynamically distributes contextual features to produce the output as $\boldsymbol{y}_n = \boldsymbol{\lambda}_n^T \boldsymbol{q}_n$. This process captures content-based and position-based interactions without producing attention maps.

Multi-query lambda layers

Lets create $|h|$ queries q_n^h, apply the same lambda λ_n to each query q_n^h, and concatenate the outputs as $y_n = \text{concat}(\lambda_n q_n^1, ..., \lambda_n q_n^{|h|})$. Now $|v| = d/|h|$.



Time Complexity: $\Theta(bnmkv) \rightarrow \Theta(bnmkd/h)$
Space Complexity: $\Theta(knm + bnkv) \rightarrow \Theta(knm + bnkd/h)$

Realization simplicity: einsum

```
def lambda_layer(queries, keys, embeddings, values):
    """Multi-query lambda layer."""
    # b: batch, n: input length, m: context length,
    # k: query/key depth, v: value depth,
    # h: number of heads, d: output dimension.
    content_lambda = einsum(softmax(keys), values, 'bmk,bmv->bkv')
    position_lambdas = einsum(embeddings, values, 'nmk,bmv->bnkv')
    content_output = einsum(queries, content_lambda, 'bhnk,bkv->bnhv')
    position_output = einsum(queries, position_lambdas, 'bhnk,bnkv->bnhv')
    output = reshape(content_output + position_output, [b, n, d])
    return output
```

Figure 3: **Pseudo-code for the multi-query lambda layer**. The position embeddings can be made to satisfy various conditions, such as translation equivariance, when computing positional lambdas (not shown).

Self-attention: multi-head attention

1) This is our input sentence*

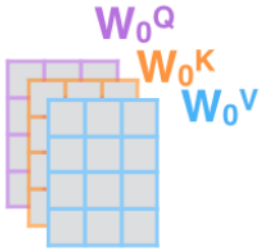
2) We embed each word*

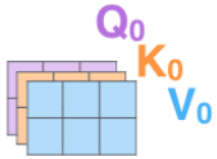
3) Split into 8 heads. We multiply X or R with weight matrices

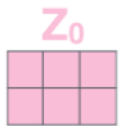
4) Calculate attention using the resulting Q/K/V matrices

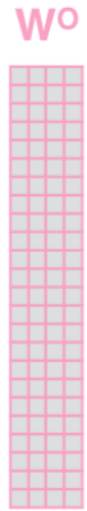
5) Concatenate the resulting Z matrices, then multiply with weight matrix W^O to produce the output of the layer

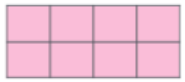
Thinking Machines



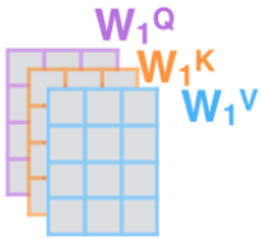


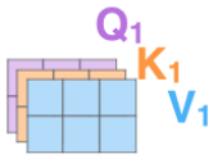


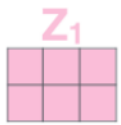




* In all encoders other than #0, we don't need embedding. We start directly with the output of the encoder right below this one



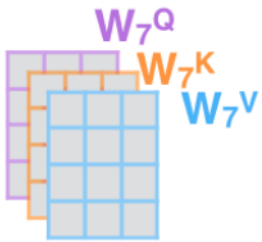


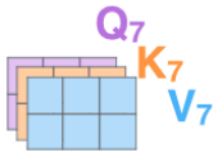


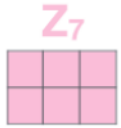
...

...

...







Lambda Convolution

The property that shifting the inputs results in an equivalent shift of the outputs is a strong inductive bias in many learning scenarios.

$E \in \mathbb{R}^{|n| \times |m| \times |k|} \quad \rightarrow \quad R \in \mathbb{R}^{|r| \times |k|} : \quad e_{nm} = r_{r(n,m)}$

```
# embeddings shape: [r, k]
embeddings = reshape(embeddings, [r, 1, 1, k])
values = reshape(values, [b, n, v, 1])
position lambdas = conv2d(values, embeddings)
# Transpose from shape [b, n, v, k] to shape [b, n, k, v]
position lambdas = transpose(position lambdas, [0, 1, 3, 2])
```

Time Complexity: $\Theta(bnmkd/h) \rightarrow \Theta(bnrkd/h)$

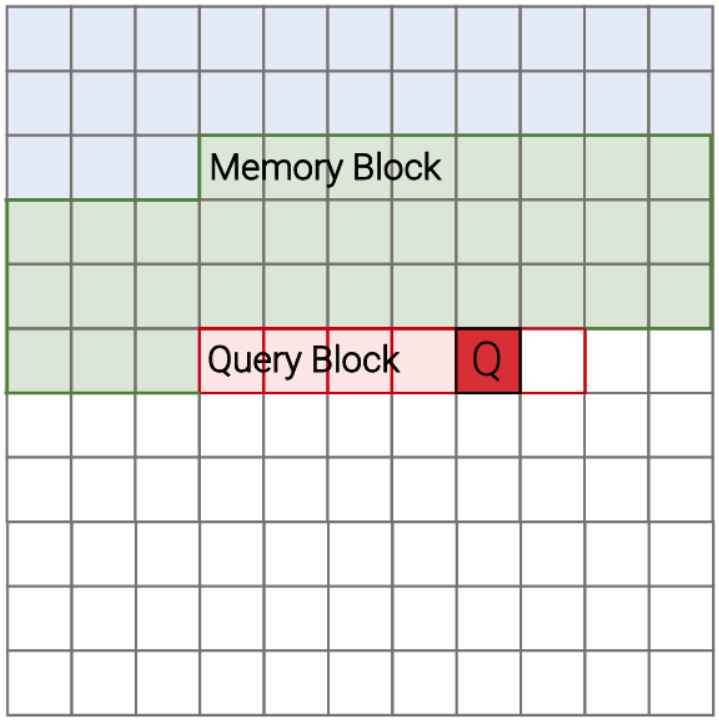
Space Complexity: $\Theta(knm + bnkd/h) \rightarrow \Theta(kr + bnkd/h)$

Lambda Convolution

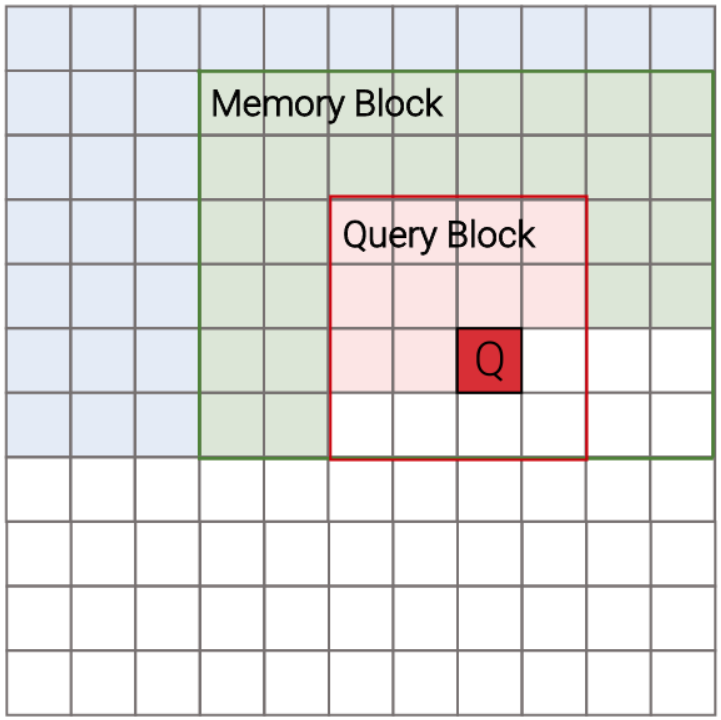
Scope size $\|m\|$	3x3	7x7	15x15	23x23	31x31	global
FLOPS (B)	5.7	6.1	7.8	10.0	12.4	19.4
Top-1 Accuracy	77.6	78.2	78.5	78.3	78.5	78.4

Table 11: **Impact of varying the scope size for positional lambdas on the ImageNet classification task**. We replace the 3x3 spatial convolutions in the *last 2 stages* of a ResNet-50 with lambda layers (input image size is 224x224). Flops significantly increase with the scope size, however we stress that larger scopes do not translate to slower latencies when using the einsum implementation (see Figure 3).

Local Self-Attention



(a) 1-dimensional local attention



(b) 2-dimensional local attention

Figure 3: Attention span in Image Transformer on a two-dimensional input.

Self-attention and Lambda layers

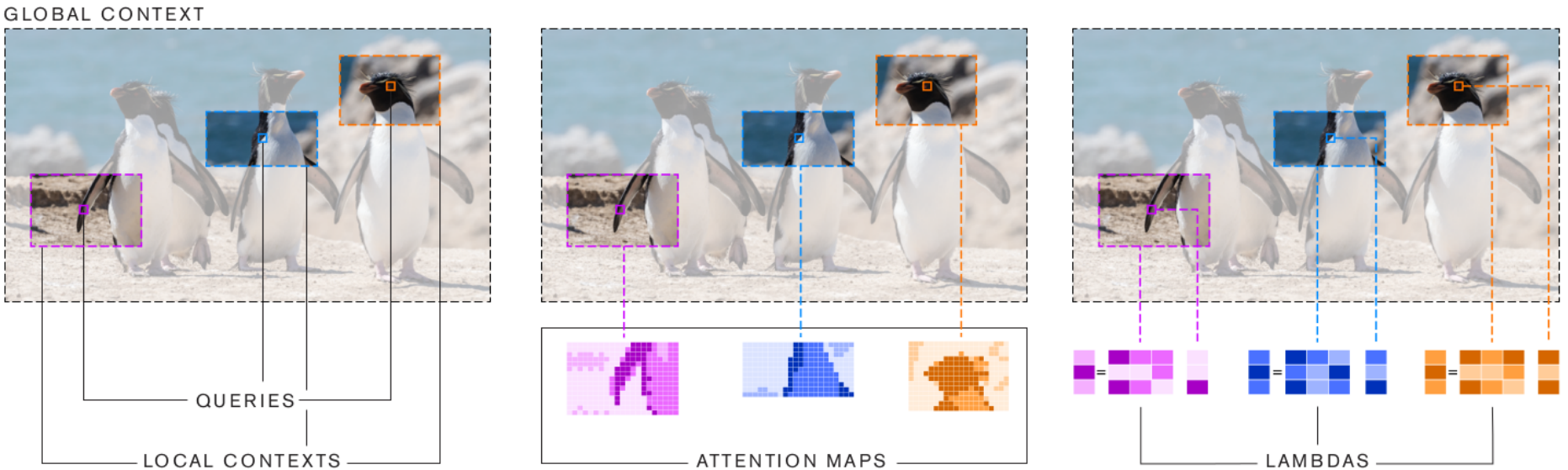
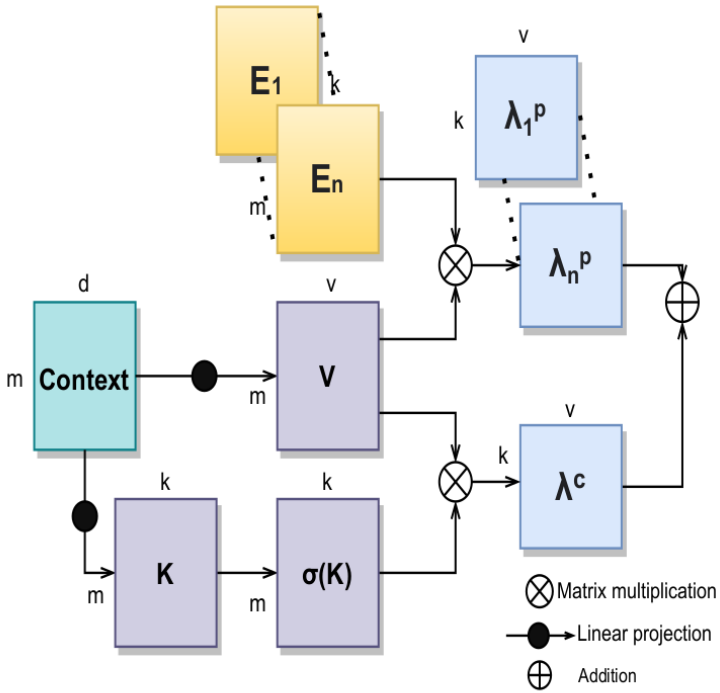


Figure 1: **Comparison between self-attention and lambda layers**. (**Left**) An example of 3 queries and their local contexts within a global context. (**Middle**) Self-attention associates each query with an attention distribution over its context. (**Right**) The lambda layer transforms each context into a linear function lambda that is applied to the corresponding query.

Intra-depth

Operation	Time complexity	Space complexity
Lambda layer ($\|u\| > 1$)	$\Theta(bnmkud/h)$	$\Theta(knmu + bnkv)$

Table 8: **Complexity for a multi-query lambda layer with intra-depth $|u|$.**



Generating lambdas from the context

Lambda Networks vs Other Attentions

Layer	Params (M)	top-1
Conv (He et al., 2016)†	25.6	76.9
Conv + channel attention (Hu et al., 2018c)†	28.1	77.6 (+0.7)
Conv + linear attention (Chen et al., 2018)	33.0	77.0
Conv + linear attention (Shen et al., 2018)	-	77.3 (+1.2)
Conv + relative self-attention (Bello et al., 2019)	25.8	77.7 (+1.3)
Local relative self-attention (Ramachandran et al., 2019)	18.0	77.4 (+0.5)
Local relative self-attention (Hu et al., 2019)	23.3	77.3 (+1.0)
Local relative self-attention (Zhao et al., 2020)	20.5	78.2 (+1.3)
Lambda layer	**15.0**	**78.4** (**+1.5**)
Lambda layer ($\|u\|$=4)	**16.0**	**78.9** (**+2.0**)

Table 3: **Comparison of the lambda layer and attention mechanisms on ImageNet classification with a ResNet50 architecture.** The lambda layer strongly outperforms attention alternatives at a fraction of the parameter cost. All models are trained in mostly similar setups (see Appendix E.2) and we include the reported improvements compared to the convolution baseline in parentheses. See Appendix B.4 for a description of the $|u|$ hyperparameter. † Our implementation.

Lambda Networks vs Other Attentions

Layer	Space Complexity	Memory (GB)	Throughput	top-1
Global self-attention	$\Theta(blhn^2)$	120	OOM	OOM
Axial self-attention	$\Theta(blhn\sqrt{n})$	4.8	960 ex/s	77.5
Local self-attention (7x7)	$\Theta(blhnm)$	-	440 ex/s	77.4
Lambda layer	$\Theta(lkn^2)$	1.9	1160ex/s	**78.4**
Lambda layer ($\|k\|$=8)	$\Theta(lkn^2)$	0.95	**1640** ex/s	77.9
Lambda layer (shared embeddings)	$\Theta(kn^2)$	**0.63**	1210 ex/s	78.0
Lambda convolution (7x7)	$\Theta(lknm)$	-	1100 ex/s	78.1

Table 4: **The lambda layer reaches higher ImageNet accuracies while being faster and more memory-efficient than self-attention alternatives.** Memory is reported assuming full precision for a batch of 128 inputs using default hyperparameters. The memory cost for storing the lambdas matches the memory cost of activations in the rest of the network and is therefore ignored. b: batch size, h: number of heads/queries, n: input length, m: context length, k: query/key depth, l: number of layers.

Lambda Networks vs EfficientNet

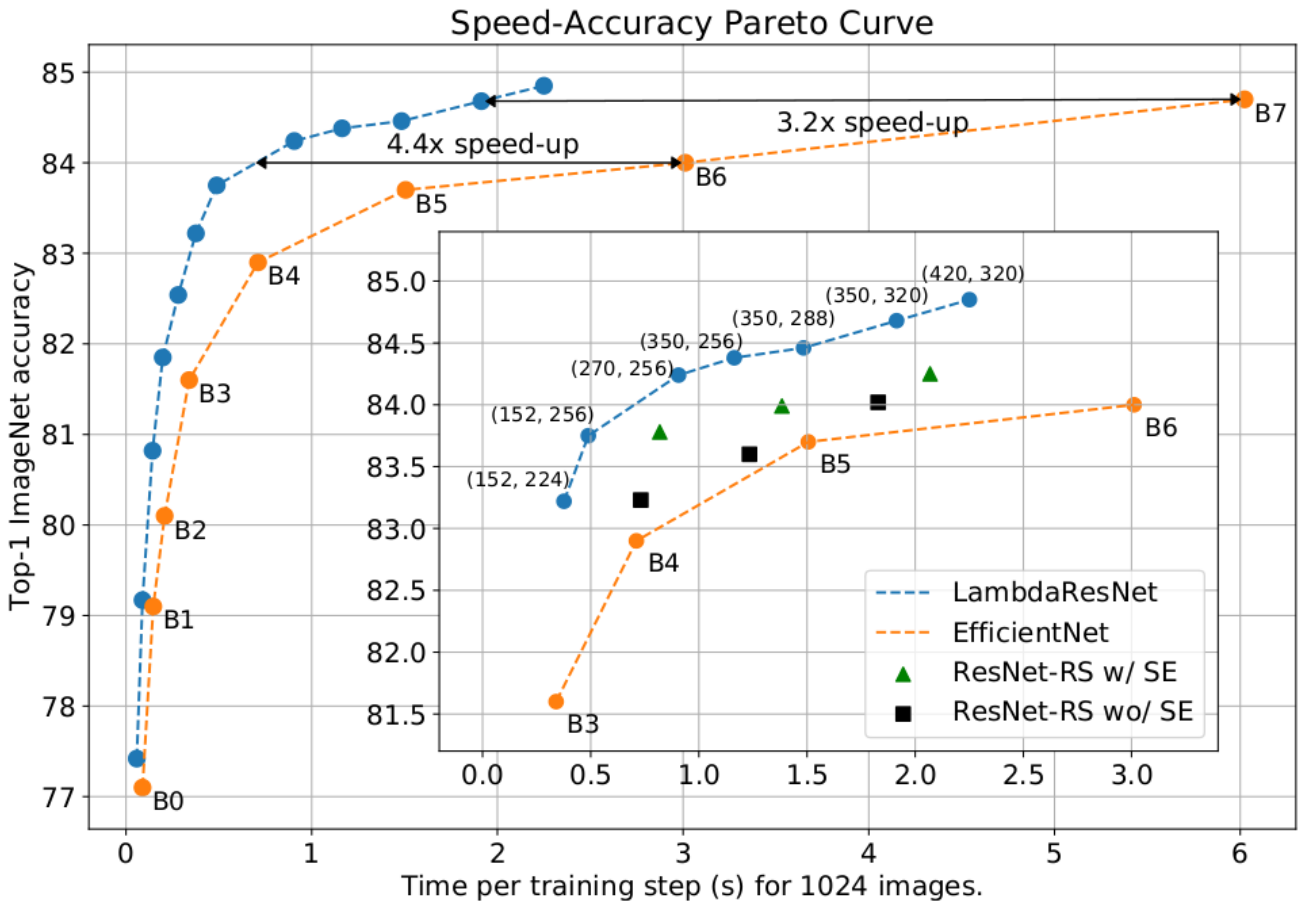


Figure 4: **Speed-accuracy comparison between LambdaResNets and EfficientNets.** When matching the training and regularization setup of EfficientNets, LambdaResNets are 3.2 - 4.4x faster than EfficientNets and 1.6 - 2.3x faster than ResNet-RS with squeeze-and-excitation. LambdaResNets are annotated with (depth, image size). Our largest LambdaResNet, LambdaResNet-420 trained at image size 320, reaches a strong 84.9% top-1 accuracy.

Lambda Networks + pseudo-labeling

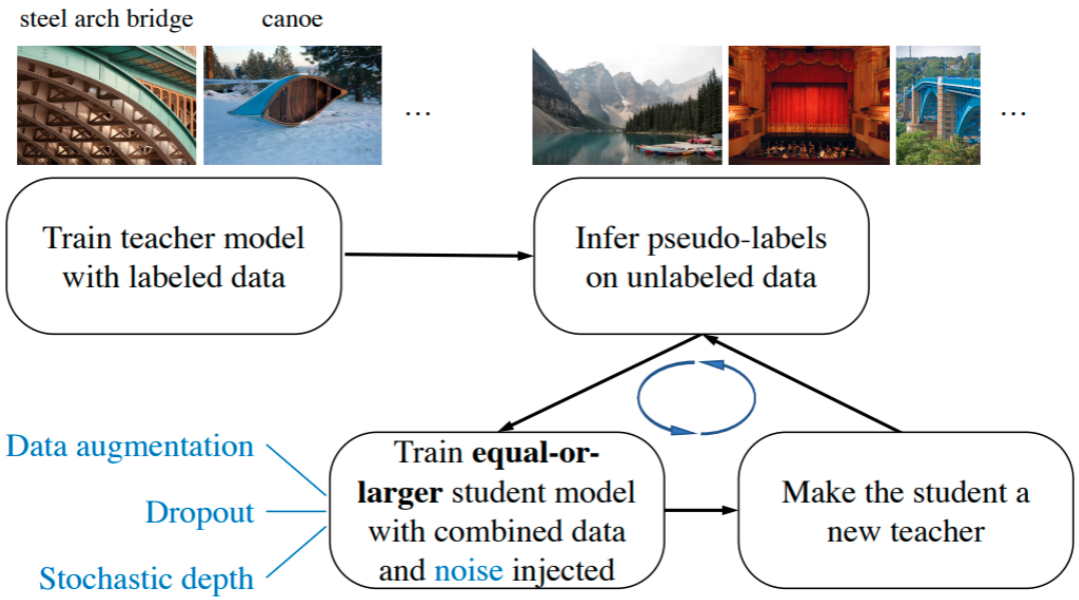


Figure 1: Illustration of the Noisy Student Training. (All shown images are from ImageNet.)

Lambda Networks + pseudo-labeling

Architecture	Params (M)	Train (ex/s)	Infer (ex/s)	ImageNet top-1
LambdaResNet-152	**51**	**1620**	**6100**	86.7
EfficientNet-B7	66	170 (9.5x)	980 (6.2x)	86.7
ViT-L/16	307	180 (9.0x)	640 (9.5x)	**87.1**

Table 5: **Comparison of models trained on extra data**. ViT-L/16 is pre-trained on JFT and fine-tuned on ImageNet at resolution 384x384, while EfficientNet and LambdaResNet are co-trained on ImageNet and JFT pseudo-labels. Training and inference throughput is shown for 8 TPUv3 cores.

Object Detection and Instance Segmentation

Backbone	AP^{bb}_{coco}	$AP^{bb}_{s/m/l}$	AP^{mask}_{coco}	$AP^{mask}_{s/m/l}$
ResNet-101	48.2	29.9 / 50.9 / 64.9	42.6	24.2 / 45.6 / 60.0
ResNet-101 + SE	48.5	29.9 / 51.5 / 65.3	42.8	24.0 / 46.0 / 60.2
LambdaResNet-101	**49.4**	**31.7** / **52.2** / **65.6**	**43.5**	**25.9** / **46.5** / **60.8**
ResNet-152	48.9	29.9 / 51.8 / 66.0	43.2	24.2 / 46.1 / 61.2
ResNet-152 + SE	49.4	30.0 / 52.3 / 66.7	43.5	24.6 / 46.8 / 61.8
LambdaResNet-152	**50.0**	**31.8** / **53.4** / **67.0**	**43.9**	**25.5** / **47.3** / **62.0**

Table 6: **COCO object detection and instance segmentation with Mask-RCNN architecture on 1024x1024 inputs**. Mean Average Precision (AP) for small, medium, large objects (s/m/l). Using lambda layers yields consistent gains across all object sizes, especially small objects.

Thank you for your attention!