LAMBDANETWORKS: MODELING LONG-RANGE INTERACTIONS WITHOUT ATTENTION

Anonymous authors
Paper under double-blind review

ABSTRACT

We present a framework for capturing long-range interactions between an input and structured contextual information (e.g. a pixel surrounded by other pixels). Our method, called the lambda layer, captures such interactions by transforming available contexts into linear functions, termed lambdas, and applying these linear functions to each input separately. Lambda layers may be implemented to model content and position-based interactions in global, local or masked contexts. As they bypass the need for expensive attention maps, lambda layers can routinely be applied to inputs of length in the thousands, enabling their applications to long sequences or high-resolution images. The resulting neural network architectures, *LambdaNetworks*, are computationally efficient and simple to implement using direct calls to operations available in modern neural network libraries. Experiments on ImageNet classification and COCO object detection and instance segmentation demonstrate that LambdaNetworks significantly outperform their convolutional and attentional counterparts while being more computationally efficient. Finally, we introduce LambdaResNets, a family of architectures that considerably improve the speed-accuracy tradeoff of image classification models. LambdaResNets reach state-of-the-art accuracies on ImageNet while being $\sim$4.5x faster than the popular EfficientNets on modern machine learning accelerators.

1 INTRODUCTION

Modeling long-range interactions is of central importance in machine learning. Attention (Bahdanau et al., 2015; Vaswani et al., 2017) has emerged as the paradigm of choice for capturing long-range interactions. However, the quadratic memory footprint of self-attention has hindered its applicability to long sequences or multidimensional inputs such as images which typically contain tens of thousands of pixels. For example, applying a single multi-head attention layer to a batch of 256 of 64x64 input images with 8 heads requires 32GB of memory, which is prohibitive in practice.

This work presents a class of layers, termed lambda layers, which provide a general framework for capturing long-range interactions between an input and a structured set of context elements. Lambda layers transform available contexts into individual linear functions, termed *lambdas*, that are directly applied to each input separately. We motivate lambda layers as a natural alternative to attention mechanisms. Whereas attention defines a similarity kernel between the input and context elements, lambda layers summarize contextual information into a fixed-size linear function, thus bypassing the need for memory-expensive attention maps. This contrast is illustrated in Figure 1.

We demonstrate the versatility of lambda layers and show that they may be implemented to capture content-based *and position-based* interactions in *global*, *local* or *masked* contexts. The resulting neural networks, *LambdaNetworks*, are computationally efficient, model long-range dependencies at a small memory cost and can therefore be routinely applied to large structured inputs such as high resolution images. We evaluate LambdaNetworks on computer vision tasks where self-attention has shown promise (Bello et al., 2019; Ramachandran et al., 2019) but has suffered from large memory costs and impractical implementations. Controlled experiments on ImageNet classification and COCO object detection and instance segmentation indicate that LambdaNetworks significantly outperform their convolutional and attentional counterparts while being more computationally efficient and much faster than the latter. Finally, we introduce LambdaResNets, a family of hybrid LambdaNetworks across different scales, which considerably improve the speed-accuracy tradeoff of im-

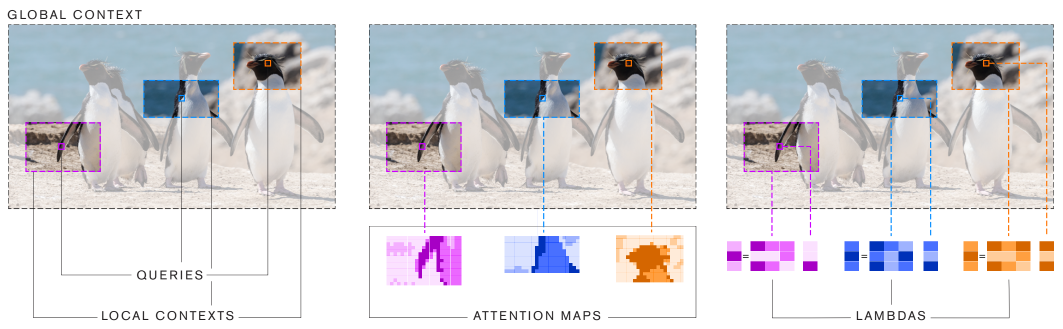

Figure 1: Comparison between attention and lambda layers. (Left) An example of 3 queries and their local contexts within a global context. (Middle) The attention operation associates each query with an attention distribution over its context. (Right) The lambda layer transforms each context into a linear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

age classification models. In particular, LambdaResNets reach state-of-the-art ImageNet accuracies while being 4.5x faster than EfficientNets.

2 Modeling long-range interactions

In this section, we formally define queries, contexts and interactions. We motivate keys as a requirement for capturing interactions between queries and their contexts and show that lambda layers arise as an alternative to attention mechanisms for capturing long-range interactions.

Notation. We denote scalars, vectors and tensors using lower-case, bold lower-case and bold upper-case letters, *e.g.,* n, $\boldsymbol{x}$ and $\boldsymbol{X}$. We denote $|n|$ the cardinality of a set whose elements are indexed by n. We denote $\boldsymbol{x}_n$ the n-th row of $\boldsymbol{X}$ and $\{\boldsymbol{x}_n\}$ the collection of its $|n|$ rows. We denote x_{ij} the $|ij|$ elements of $\boldsymbol{X}$. When possible, we adopt the terminology of self-attention to ease readability and highlight differences.

Defining queries, contexts and interactions. Let $\mathcal{Q} = \{(\boldsymbol{q}_n, n)\}$ and $\mathcal{C} = \{(\boldsymbol{c}_m, m)\}$ denote structured collections of vectors, respectively referred to as the *queries* and the *context*. Each query $(\boldsymbol{q}_n, n)$ is characterized by its content $\boldsymbol{q}_n \in \mathbb{R}^{|k|}$ and *position* n. Similarly, each context element $(\boldsymbol{c}_m, m)$ is characterized by its *content* $\boldsymbol{c}_m$ and its position m in the context. The (n, m) pair may refer to any type of pairwise relation between structured elements. For example, it could refer to the 2D relative distance between pixels arranged in a two-dimensional grid or to edge relations between nodes in a graph.

We consider the general problem of mapping a query $(\boldsymbol{q}_n, n)$ to an output vector $\boldsymbol{y}_n \in \mathbb{R}^{|v|}$ given the context $\mathcal{C}$ with a function $\boldsymbol{F} : ((\boldsymbol{q}_n, n), \mathcal{C}) \mapsto \boldsymbol{y}_n$. Such a function may act as a layer in a neural network when processing structured inputs. We refer to $(\boldsymbol{q}_n, \boldsymbol{c}_m)$ interactions as *content-based* and $(\boldsymbol{q}_n, (n, m))$ interactions as *position-based*. Additionally, we say that $\boldsymbol{F}$ captures *global* interactions when the output $\boldsymbol{y}_n$ depends on all $(\boldsymbol{q}_n, \boldsymbol{c}_m)$ (or $(\boldsymbol{q}_n, (n, m))$) interactions and *local* when only a restricted smaller context around n is considered. Finally, these interactions are defined as *dense* if they include all $|m|$ elements in the context and *sparse* otherwise.

Introducing keys to capture long-range interactions. In the context of deep learning, we prioritize fast batched linear operations and choose our interactions to be captured by dot-product operations. This motivates introducing vectors that can interact with the queries via a dot-product operation and therefore have the same dimension as the queries. In particular, content-based interactions $(\boldsymbol{q}_n, \boldsymbol{c}_m)$ require a $|k|$-dimensional vector that depends on $\boldsymbol{c}_m$, commonly referred to as the key $\boldsymbol{k}_m$. Conversely, position-based interactions $(\boldsymbol{q}_n, (n, m))$ require a positional embedding $\boldsymbol{e}_{nm} \in \mathbb{R}^{|k|}$, sometimes called a relative key (Shaw et al., 2018). As the query/key depth $|k|$ and context spatial dimension $|m|$ are not in the output $\boldsymbol{y}_n \in \mathbb{R}^{|v|}$, these dimensions need to be contracted as part of the layer computations. *Every layer capturing long-range interactions can therefore be characterized based on whether it contracts the query depth or the context positions first.*

Table 1: Hyperparameter, parameters and quantities of interest describing the lambda layer.

Name	Type	Description
$\lvert k\rvert$, $\lvert v\rvert$	hyperparameter	key/query depth, value depth
$\lvert u\rvert$		intra-depth
$\boldsymbol{W}_Q \in \mathbb{R}^{d\times\lvert k\rvert}$	parameter	a tensor that linearly projects the inputs
$\boldsymbol{W}_K \in \mathbb{R}^{d\times\lvert k\rvert\times\lvert u\rvert}$		a tensor that linearly projects the context
$\boldsymbol{W}_V \in \mathbb{R}^{d\times\lvert v\rvert\times\lvert u\rvert}$		a tensor that linearly projects the context
$\boldsymbol{E}_{nm} \in \mathbb{R}^{\lvert k\rvert\times\lvert u\rvert}$		a positional embedding for the relation (n, m).
$\boldsymbol{X} \in \mathbb{R}^{\lvert n\rvert\times d}$	input	the inputs
$\boldsymbol{C} \in \mathbb{R}^{\lvert m\rvert\times d}$		the context
$\boldsymbol{Q} = \boldsymbol{X}\boldsymbol{W}_Q \in \mathbb{R}^{\lvert n\rvert\times\lvert k\rvert}$	activation	the queries
$\boldsymbol{K} = \boldsymbol{C}\boldsymbol{W}_K \in \mathbb{R}^{\lvert m\rvert\times\lvert k\rvert\times\lvert u\rvert}$		the keys
$\boldsymbol{V} = \boldsymbol{C}\boldsymbol{W}_V \in \mathbb{R}^{\lvert m\rvert\times\lvert v\rvert\times\lvert u\rvert}$		the values
$\bar{\boldsymbol{K}} = \text{softmax}_m(\boldsymbol{K})$		the normalized keys
$\boldsymbol{\lambda}^c = \sum_m \boldsymbol{K}_m \boldsymbol{V}_m^T \in \mathbb{R}^{\lvert k\rvert\times\lvert v\rvert}$		*content* lambda shared across all queries
$\boldsymbol{\lambda}_n^p = \sum_m \boldsymbol{E}_{nm} \boldsymbol{V}_m^T \in \mathbb{R}^{\lvert k\rvert\times\lvert v\rvert}$		*position* lambda for the n-th query
$\boldsymbol{\lambda}_n = \boldsymbol{\lambda}^c + \boldsymbol{\lambda}_n^p \in \mathbb{R}^{\lvert k\rvert\times\lvert v\rvert}$		lambda for the n-th query

Attentional interactions. Contracting the query depth first creates a similarity kernel (the attention map) between the query and context elements and is known as the attention operation. This mechanism can be viewed as addressing a differentiable memory which motivates the query, key, value terminology. As the number of context positions $|m|$ grows larger and the input and output dimensions $|k|$ and $|v|$ remain fixed, one may hypothesize that computing attention maps become wasteful, given that the layer output is a vector of comparatively small dimension $|v| \ll |m|$.

Lambda interactions. Instead, it may be more efficient to simply map each query to its output via a linear function as $\boldsymbol{y}_n = F((\boldsymbol{q}_n, n), \mathcal{C}) = \boldsymbol{\lambda}(\mathcal{C}, n)(\boldsymbol{q}_n)$ for some *linear* function $\boldsymbol{\lambda}(\mathcal{C}, n)$. In this scenario, the context is aggregated into a fixed-size linear function $\boldsymbol{\lambda}_n = \boldsymbol{\lambda}(\mathcal{C}, n)$. Each $\boldsymbol{\lambda}_n$ acts as a small linear function that exist independently of the context (once computed) and is discarded after being applied to its associated query $\boldsymbol{q}_n$. This mechanism is reminiscent of functional programming and λ-calculus which motivates the lambda terminology.

3 Lambda layers

A *lambda layer* takes the inputs $\boldsymbol{X} \in \mathbb{R}^{|n|\times d_{in}}$ and the context $\boldsymbol{C} \in \mathbb{R}^{|m|\times d_c}$ as input and generates linear function lambdas that are then applied to the queries, yielding outputs $\boldsymbol{Y} \in \mathbb{R}^{|n|\times d_{out}}$. Note that we may have $\boldsymbol{C} = \boldsymbol{X}$, as is the case for self-attention. Without loss of generality, we assume $d_{in} = d_c = d_{out} = d$. In the rest of this paper, we focus on a specific instance of a lambda layer and show that it enables dense long-range content and position-based interactions without materializing attention maps.

3.1 The lambda layer: Transforming contexts into linear functions

We first describe our lambda layer in the context of a *single query* $(\boldsymbol{q}_n, n)$. As we wish to generate a linear function lambda $\mathbb{R}^{|k|} \to \mathbb{R}^{|v|}$, we interchangeably refer to $\mathbb{R}^{|k|\times|v|}$ matrices as functions. Hyperparameters, parameters and other quantities of interest of our lambda layer are presented in Table 1.

Generating the contextual lambda function. Our lambda layer first computes *keys* and *values* by linearly projecting the context, and keys are normalized across context positions via a softmax operation yielding normalized keys $\bar{\boldsymbol{K}}$. The $\boldsymbol{\lambda}_n$ function is obtained by summing contributions from the context as

$$\boldsymbol{\lambda}_n = \sum_m (\bar{\boldsymbol{K}}_m + \boldsymbol{E}_{nm})\boldsymbol{V}_m^T \in \mathbb{R}^{|k|\times|v|} \tag{1}$$

The *content lambda* $\boldsymbol{\lambda}^c = \sum_m \bar{\boldsymbol{K}}_m \boldsymbol{V}_m^T$ is invariant to permutation of the context elements, shared across all query positions n and encodes how to transform the query $\boldsymbol{q}_n$ solely based on the context content. In contrast, the *position lambda* $\boldsymbol{\lambda}_n^p = \sum_m \boldsymbol{E}_{nm} \boldsymbol{V}_m^T$ encodes how to transform the query content $\boldsymbol{q}_n$ based on the content $\boldsymbol{c}_m$ *and positions* (n, m), enabling modeling structured inputs such as images.

Applying lambda to its query. The input $\boldsymbol{x}_n$ is then transformed into a *query* $\boldsymbol{q}_n = \boldsymbol{W}_Q \boldsymbol{x}_n$ and the output of the lambda layer is obtained as

$$\boldsymbol{y}_n = \boldsymbol{\lambda}_n^T \boldsymbol{q}_n = (\boldsymbol{\lambda}^c + \boldsymbol{\lambda}_n^p)^T \boldsymbol{q}_n \in \mathbb{R}^{|v|}. \tag{2}$$

Lambda interpretation. The columns of the $\boldsymbol{\lambda}_n \in \mathbb{R}^{|k| \times |v|}$ matrix can be viewed as a fixed-size set of $|k|$ $|v|$-dimensional contextual features. These contextual features are aggregated from the context's content (content-based interactions) and structure (position-based interactions). Applying the lambda linear function dynamically distributes these contextual features to produce the output as $\boldsymbol{y}_n = \sum_k q_{nk} \boldsymbol{\lambda}_{nk}$. This process captures *dense content and position-based long-range interactions* without producing attention maps.

Normalization. One may modify Equations 1 and 2 to include non-linearities or normalization operations. Our experiments indicate that applying batch normalization (Ioffe & Szegedy, 2015) after computing the queries and the values is helpful.

3.2 Lambda layers with structured contexts

This section presents how to adapt our lambda layer to *structured* contexts, such as *relative* and *local* contexts. We discuss *masked* contexts and their applications in the Appendix A.

Translation equivariance Translation equivariance is a strong inductive bias in many learning scenarios. The content-based interactions are permutation equivariant and hence already translation equivariant. We obtain translation-equivariance in position interactions by ensuring that the position embeddings satisfy $\boldsymbol{E}_{nm} = \boldsymbol{E}_{t(n)t(m)}$ for any translation t. In practice, we define a tensor of *relative* position embeddings $\boldsymbol{R} \in \mathbb{R}^{|k| \times |r| \times |u|}$, where r indexes the possible relative positions for all (n, m) pairs, and reindex it into $\boldsymbol{E} \in \mathbb{R}^{|k| \times |n| \times |m| \times |u|}$ such that $\boldsymbol{E}_{nm} = \boldsymbol{R}_{r(n,m)}$.

Lambda convolution Despite the benefits of long-range interactions, locality remains a strong inductive bias in many tasks. Using global contexts may prove noisy or excessive from a computational standpoint. It may therefore be useful to restrict the scope of position interactions to a *local* neighborhood around the query position n as is the case for local self-attention and convolutions. This can be done by zeroing out the position embeddings for context positions m outside of the desired scope. However, this strategy remains costly for large values of $|m|$ since the computations still occur (they are only being zeroed out).

In the case where the context is arranged on a multidimensional grid, we can generate *positional lambdas from local contexts by using a regular convolution* that treats the v dimension in $\boldsymbol{V}$ as an *extra spatial dimension*. For example, let's assume we want to generate positional lambdas with local scope size $|r|$ on *1d* sequences. The relative position embedding tensor $\boldsymbol{R} \in \mathbb{R}^{|r| \times |u| \times |k|}$ can be reshaped to $\bar{\boldsymbol{R}} \in \mathbb{R}^{|r| \times 1 \times |u| \times |k|}$ and used as the kernel of a *2d* convolution to compute the desired position lambda as

$$\boldsymbol{\lambda}_{bnvk} = \text{conv2d}(\boldsymbol{V}_{bnvu}, \bar{\boldsymbol{R}}_{r1uk}). \tag{3}$$

We term this operation the *lambda convolution*. As the computations are now restricted to a local scope, the lambda convolution obtains *linear time and memory complexities with respect to the input length*. The lambda convolution is readily usable with additional functionalities such as dilation and striding and enjoys highly optimized implementations on specialized hardware accelerators (Nickolls & Dally, 2010; Jouppi et al., 2017). This is in stark contrast to implementations of local self-attention (Parmar et al., 2018; Ramachandran et al., 2019) which require materializing feature patches of overlapping query and memory blocks, increasing memory consumption and latency (see Table 4).

Table 2: **The lambda layer captures content and position-based interactions between queries and contexts without materializing per-example attention maps.** b: batch size, h: number of heads/queries, n: input length, m: context length, k: query/key depth, d: dimension output.

	Content interactions		Position interactions	
	Time	Space	Time	Space
Attention layer	$\Theta(bnm(hk+d))$	$\Theta(bhnm)$	$\Theta(bnm(hk+d))$	$\Theta(bhnm)$
Lambda layer	$\Theta(bmkd/h)$	-	$\Theta(bnmkd/h)$	$\Theta(knm+bnkd/h)$

3.3 Reducing complexity with multiquery lambdas.

Complexity analysis. For a batch of $|b|$ elements, each containing $|n|$ inputs, the number of arithmetic operations and memory footprint required to apply our lambda layer are respectively $\Theta(bnmkv)$ and $\Theta(knm + bnkv)$. We still have a quadratic memory footprint with respect to the input length due to the $\boldsymbol{E}_{nm}$ parameters that capture position-based interactions. However this quadratic term does not scale with the batch size as is the case with the attention operation which produces *per-example* attention maps. In practice, the hyperparameter $|k|$ is set to a small value (such as $|k|$=16) and we can process large batches of large inputs in cases where attention cannot (see Table 4).

Multiquery lambdas reduce complexity. Recall that the lambdas map queries $\boldsymbol{q}_n \in \mathbb{R}^k$ to outputs $\boldsymbol{y}_n \in \mathbb{R}^d$. As presented in Equation 2, this implies that $|v|$=d. Small values of $|v|$ may therefore act as a bottleneck on the feature vector $\boldsymbol{y}_n$ but larger output dimensions $|v|$ can incur an excessively large computational cost given our $\Theta(bnmkv)$ and $\Theta(bnkv + knm)$ time and space complexities.

We propose to decouple the time and space complexities of our lambda layer from the output dimension d. Rather than imposing $|v|$=d, we create $|h|$ queries $\{\boldsymbol{q}_n^h\}$, apply the same lambda function $\boldsymbol{\lambda}_n$ to each query $\boldsymbol{q}_n^h$, and concatenate the outputs as $\boldsymbol{y}_n = \text{concat}(\boldsymbol{\lambda}_n \boldsymbol{q}_n^1, \cdots, \boldsymbol{\lambda}_n \boldsymbol{q}_n^{|h|})$.

We refer to this operation as a *multiquery lambda* layer as each lambda is applied to $|h|$ queries. This can also be interpreted as constraining the lambda to a smaller block matrix with $|h|$ equal repeated blocks. We now have d=$|hv|$ and our time and space complexities become $\Theta(bnmkd/h)$ and $\Theta(knm + bnkd/h)$.

We note that while this resembles the multihead or multiquery (Shazeer, 2019) attention formulation, the motivation is different. Using multiple queries in the attention operation increases representational power and complexity. In our case, using multiquery lambdas reduces complexity and representational power. Table 2 compares time and space complexities of the multiquery lambda layer and the multihead attention operation.

Batched multiquery implementation with einsum. The multiquery lambda layer can be applied on a batch of inputs using einsum[1] as:

$$\begin{aligned}
\boldsymbol{\lambda}^c_{bkv} &= einsum(\bar{\boldsymbol{K}}_{bmku}, \boldsymbol{V}_{bmvu}) \\
\boldsymbol{\lambda}^p_{bnkv} &= einsum(\boldsymbol{E}_{knmu}, \boldsymbol{V}_{bmvu}) \\
\boldsymbol{Y}^c_{bnhv} &= einsum(\boldsymbol{Q}_{bnhk}, \boldsymbol{\lambda}^c_{bkv}) \\
\boldsymbol{Y}^p_{bnhv} &= einsum(\boldsymbol{Q}_{bnhk}, \boldsymbol{\lambda}^p_{bnkv}) \\
\boldsymbol{Y}_{bnhv} &= \boldsymbol{Y}^c_{bnhv} + \boldsymbol{Y}^p_{bnhv}
\end{aligned} \tag{4}$$

and a reshaping operation $\boldsymbol{Y}_{bnhv} \to \boldsymbol{Y}_{bnd}$. In the special case $|u| = 1$, we work with the squeezed tensors and the indice u can be removed from the einsum equations. Local positional lambdas may instead be obtained via the lambda convolution as in Equation 3.

[1]The einsum operation denotes general contractions between tensors of arbitrary dimensions. It is numerically equivalent to broadcasting its inputs to share the union of their dimensions, multiplying element-wise and summing across all dimensions not specified in the output. We describe the shape of a tensor by simply concatenating its dimensions. For example, a batch of b sequences of n d-dimensional vectors has shape bnd.

Table 3: **Comparison of the lambda layer and attention mechanisms on ImageNet classification with a ResNet50 architecture.** The lambda layer strongly outperforms alternatives at a fraction of the parameter cost. We include the reported improvements compared to the ResNet50 baseline in subscript to account for training setups that are not directly comparable. †: Our implementation.

Layer	Params (M)	top-1
Conv (He et al., 2016)†	25.6	$76.9_{+0.0}$
Conv + channel attention (Hu et al., 2018b)†	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**	$\mathbf{78.4}_{+1.5}$
Lambda layer ($\vert u \vert$=4)	**16.0**	$\mathbf{78.9}_{+2.0}$

Table 4: **The lambda layer reaches higher accuracies while being faster and more memory-efficient than self-attention alternatives.** ImageNet classification at resolution 224x224 with a ResNet50 architecture where 3x3 convolutions are replaced by the attention operation or the lambda layer. We ignore the memory complexity for storing lambdas $\Theta(bnkd/h)$ since this term matches the cost of storing activations in the rest of the network. b: batch size, h: number of heads/queries, n: input length, m: context length, k: query/key depth, l: number of layers.

Layer	Complexity	Memory (GB)	Throughput	top-1
Global self-attention	$\Theta(blhn^2)$	120	OOM	OOM
Axial self-attention	$\Theta(blhn\sqrt{n})$	4.8	960ex/s	77.5
Local self-attention (7x7)	$\Theta(blhnm)$	-	440ex/s	77.4
Lambda layer	$\Theta(lkn^2)$	0.96	1160ex/s	**78.4**
Lambda layer ($\vert k \vert$=8)	$\Theta(lkn^2)$	0.48	**1640**ex/s	77.9
Lambda layer (shared embeddings)	$\Theta(kn^2)$	0.31	1210ex/s	78.0
Lambda convolution (7x7)	$\Theta(lknm)$	-	1100ex/s	78.1

4 Related Work

While it has not been explicitly stated, the abstraction of transforming available contexts into linear functions that are applied to queries is quite general and therefore encompasses many previous works. Closest to our work are channel and linear attention mechanisms which can be cast as less flexible specific instances of *content-only* lambda interactions. Lambda layers formalize and extend such approaches to consider both content-based *and position-based* interactions, which enables their use as a stand-alone layer on highly structured inputs such as images. Rather than attempting to closely approximate attention maps as is the case in linear attention formulations, the lambda abstraction shifts the focus to the design of efficient contextual lambda functions. This enables more flexible non-linearity and normalization schemes and leads to multiquery lambdas. Controlled experiments demonstrate that lambda layers significantly outperform regular and linear attention alternatives while being more computationally efficient. We discuss related work in details in the Appendix B.

5 Experiments

In subsequent experiments, we test LambdaNetworks on standard large-scale high resolution computer vision benchmarks: ImageNet image classification task (Deng et al., 2009), COCO object detection and instance segmentation (Lin et al., 2014). The visual domain is well-suited to showcase the flexibility of lambda layers since i) the memory footprint of self-attention becomes problematic for high-resolution imagery and ii) images are highly structured, making position-based interactions

Table 5: LambdaResNets improve upon the parameter-efficiency of large EfficientNets.

Architecture	Params (M)	top-1
EfficientNet-B6	43	84.0
LambdaResNet152	**35**	84.0
LambdaResNet200	42	**84.3**

Table 6: LambdaResNets improve upon the flops-efficiency of large EfficientNets.

Architecture	Flops (G)	top-1
EfficientNet-B6	38	**84.0**
LambdaResNet-270	**34**	**84.0**

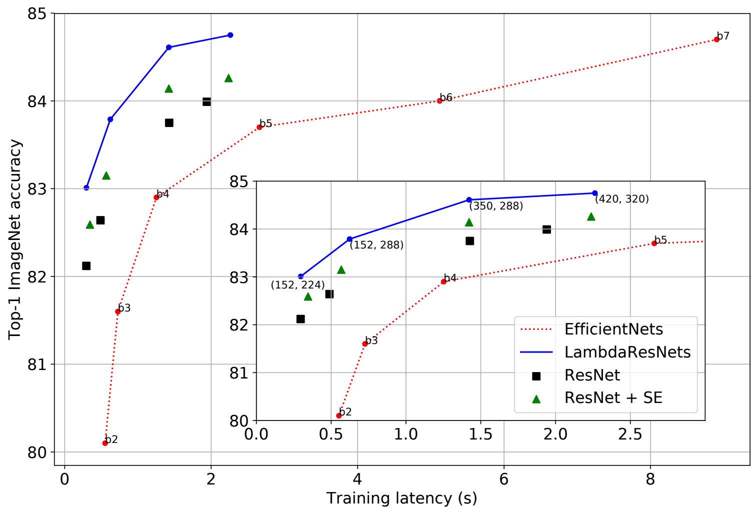

Figure 2: LambdaResNets are ∼4.5x faster than EfficientNets and substantially improve the speed-accuracy tradeoff of image classification models[3]across different (depth, image size) scales. All LambdaResNets are trained with the same hyperparameters, in contrast to EfficientNets, which use different dropout probabilities and augmentation strengths

crucial. We construct LambdaResNets by replacing the 3x3 convolutions in the ResNet architecture (He et al., 2016). Unless specified otherwise, all lambda layers use $|k|$=16, $|h|$=4 and $|u|$=1 with a scope size of $|m|$=23x23 and are implemented using the einsum implementation (Equation 4). All experiments are implemented with Tensorflow and code will be open-sourced upon publication. Experimental details can be found in the Appendix C.

LambdaNetworks outperform convolutions and attentional counterparts. In Table 3, we perform controlled experiments to compare LambdaNetworks against a) the baseline ResNet50, b) channel attention (*Squeeze-and-Excitation* or *SE*) and c) prior works that use self-attention to complement or replace the 3x3 convolutions in the ResNet50. The lambda layer strongly outperforms these approaches at a fraction of the parameter cost and notably obtains a +0.8% improvement over channel attention. The lambda layer also compares favorably against the linear attention mechanisms used in Shen et al. (2018); Chen et al. (2018).

In Table 4, we compare lambda layers against self-attention and present their throughputs, memory complexities (specifically the nm term) and ImageNet accuracies. Our results highlight the weaknesses of self-attention: self-attention cannot model global interactions due to large memory costs, axial self-attention is still memory expensive and local self-attention is prohibitively slow. In contrast, the lambda layer can capture global interactions on high-resolution images and obtains a $+1.0\%$ improvement over local self-attention while being almost 3x faster. Additionally, positional embeddings can be shared across lambda layers to further reduce memory requirements, at

[3] Ridnik et al. (2020) and Zhang et al. (2020) report high ImageNet accuracies while being up to 2x faster than EfficientNets on GPUs. We will add GPU latencies in a future draft to rigorously compare against these works. Since LambdaResNets are ∼4.5x faster than EfficientNets, we expect LambdaResNets to be much faster than these architectures as well.

Table 7: **COCO object detection and instance segmentation with Mask-RCNN architecture on 1024x1024 inputs**. Mean Average Precision (AP) is reported at three IoU thresholds and for small, medium, large objects (s/m/l).

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**

a minimal degradation cost. Finally, the lambda convolution has linear memory complexity, which becomes practical for very large images as seen in detection or segmentation.

Model ablations We perform several ablations and validate the importance of positional interactions, long-range interactions and flexible normalization schemes in the Appendix E. Table 13 presents the impact of the query depth $|k|$, number of heads $|h|$ and intra depth $|u|$ on performance. Our experiments indicate that the lambda layer outperforms convolutional and attentional baselines for a wide range of hyperparameters, demonstrating the robustness of the method. The lambda layer outperforms local self-attention when controlling for the scope size (78.1% vs 77.4% for $|m|$=7x7), suggesting that the benefits of the lambda layer go beyond improved speed and scalability.

LambdaResNets significantly improve the speed-accuracy tradeoff of ImageNet classification. In the Appendix D, we study and motivate hybrid LambdaNetwork architectures as a mean to maximize the speed-accuracy tradeoff of LambdaNetworks. The resulting hybrid LambdaResNets architectures have increased representational power at a negligible decrease in throughput compared to their vanilla ResNet counterparts. We construct hybrid LambdaResNets across various model scales by jointly scaling the depth from 50 to 420 layers and the image size from 224 to 320. Figure 2 presents the speed-accuracy curve of LambdaResNets compared to ResNets with or without channel attention and the popular EfficientNets (Tan & Le, 2019). LambdaResNets outperform the baselines across all depth and image scales with the largest LambdaResNet reaching a state-of-the-art accuracy of 84.8. Most remarkably, LambdaResNets are $\sim$4.5x faster than EfficientNets when controlling for the accuracy and significantly improve the speed-accuracy Pareto curve of image classification.

Computational efficiency. In Table 5 and Table 6, we find that it is also possible to construct LambdaResNets to improve upon the parameter and flops efficiency of large EfficientNets. These results are significant because EfficientNets were specifically designed by neural architecture search (Zoph & Le, 2017) to minimize computational costs using highly computationally efficient depthwise convolutions. In Appendix F, we show that lambda layers improve the computational-accuracy tradeoff of mobilenet architectures (Howard et al., 2017). These results suggest that lambda layers may be well suited for use in resource constrained scenarios such as embedded vision applications.

Object Detection and Instance Segmentation Lastly, we evaluate LambdaResNets on the COCO object detection and instance segmentation tasks using a Mask-RCNN architecture (He et al., 2017). Using lambda layers yields consistent gains at all IoU thresholds and all object scales, especially the harder to locate small objects. This indicates that lambda layers are also competitive for more complex visual tasks that require localization information.

6 Discussion

How do lambda layers compare to the attention operation? Similarly to attention, lambda layers can be implemented to model interactions between inputs and global, local or masked contexts. However, lambda layers scale favorably compared to self-attention. Vanilla Transformers using self-attention have $\Theta(blhn^2)$ spatial complexity, whereas LambdaNetworks have $\Theta(lkn^2)$ spatial complexity (or $\Theta(kn^2)$ when sharing positional embeddings across layers). This enables the use of lambda layers at higher-resolution and on large batch sizes. Additionally, the lambda convolution (the local version of the lambda layer) enjoys a simpler and faster implementation than local self-attention. Finally, our ImageNet experiments show that lambda layers outperforms self-attention, demonstrating that the benefits of lambda layers go beyond improved speed and scalability. In contrast to the attention operation, lambda layers may not capture fine-grained interactions or enable pointing to specific elements in the context such as in PointerNetworks (Vinyals et al., 2015).

How are lambda layers different than linear attention mechanisms? Linear attention mechanisms typically attempt to approximate the attention operation by leveraging a low-rank factorization of the attention maps. Such approaches do not consider position-based interactions and may be unnecessarily constrained in trying to closely approximate an attention kernel. In contrast, lambda layers can model *position-based* interactions which are crucial to model highly structured inputs such as images (see Table 10 in Appendix E). As the aim is not to approximate an attention kernel, lambda layers allow for more flexible non-linearities and normalizations which we also find beneficial (see Table 12 in Appendix E). Finally, we propose multiquery lambdas as a means to reduce complexity compared to the multihead (or single head) formulation typically used in linear attention works. See Appendix B for a detailed discussion of linear attention.

How to best use lambda layers in the visual domain? The improved scalability, speed and ease of implementation of lambda layers compared to global or local attention makes them a strong candidate for use in the visual domain. Our ablations demonstrate that lambda layers are most beneficial in the intermediate and low-resolution stages of vision architectures. In particular, hybrid LambdaResNets which employ a few lambda layers in these stages show significant performance improvements at a negligible increase in latency on modern machine learning accelerators.

Generality of lambda layers. While this work focuses on static 2D image recognition, we note that lambda layers can be instantiated to model interactions on structures as diverse as graphs, time series, spatial lattices, etc. We anticipate that lambda layers will be helpful in more modalities. We discuss masked contexts and auto-regressive tasks in the Appendix A.

Conclusion. We propose a new class of layers, termed lambda layers, which provide a general and scalable framework for capturing structured long-range interactions between inputs and their contexts. Lambda layers summarize available contexts into fixed-size linear functions, lambdas, that are directly applied to their associated queries. The resulting neural networks, LambdaNetworks, are simple to implement, computationally efficient and capture long-range dependencies at a small memory cost, enabling their application to large structured inputs such as high-resolution images. Extensive experiments on computer vision tasks showcase their versatility and superiority over convolutional and attentional networks. Most notably, we introduce LambdaResNets which reach state-of-the-art ImageNet accuracies and significantly improve the speed-accuracy tradeoff of image classification models.

REFERENCES

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In *International Conference on Learning Representations*, 2015.

Irwan Bello, Barret Zoph, Ashish Vaswani, Jonathon Shlens, and Quoc V. Le. Attention augmented convolutional networks. *CoRR*, abs/1904.09925, 2019. URL `http://arxiv.org/abs/1904.09925`.

Iz Beltagy, Matthew E. Peters, and Arman Cohan. Longformer: The long-document transformer. 2020.

Denny Britz, Melody Y. Guan, and Minh-Thang Luong. Efficient attention using a fixed-size memory representation. *CoRR*, abs/1707.00110, 2017. URL `http://arxiv.org/abs/1707.00110`.

Yunpeng Chen, Yannis Kalantidis, Jianshu Li, Shuicheng Yan, and Jiashi Feng. A^2-nets: Double attention networks. *CoRR*, abs/1810.11579, 2018. URL `http://arxiv.org/abs/1810.11579`.

Rewon Child, Scott Gray, Alec Radford, and Sutskever Ilya. Generating long sequences with sparse transformers. *arXiv preprint arXiv:1904.10509*.

Krzysztof Choromanski, Valerii Likhosherstov, David Dohan, Xingyou Song, Andreea Gane, Tamas Sarlos, Peter Hawkins, Jared Davis, Afroz Mohiuddin, Lukasz Kaiser, David Belanger, Lucy Colwell, and Adrian Weller. Rethinking attention with performers. 2020.

Ekin D. Cubuk, Barret Zoph, Jonathon Shlens, and Quoc V. Le. Randaugment: Practical automated data augmentation with a reduced search space. 2019.

Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc Le, and Ruslan Salakhutdinov. Transformer-xl: Attentive language models beyond a fixed-length context. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, 2019. doi: 10.18653/v1/P19-1285. URL `https://www.aclweb.org/anthology/P19-1285`.

Alexandre de Brébisson and Pascal Vincent. A cheap linear attention mechanism with fast lookups and fixed-size representations. 2016.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, 2009.

David Ha, Andrew M. Dai, and Quoc V. Le. Hypernetworks. *CoRR*, abs/1609.09106, 2016. URL `http://arxiv.org/abs/1609.09106`.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2016.

Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pp. 2980–2988, 2017.

Tong He, Zhi Zhang, Hang Zhang, Zhongyue Zhang, Junyuan Xie, and Mu Li. Bag of tricks for image classification with convolutional neural networks. 2018.

Jonathan Ho, Nal Kalchbrenner, Dirk Weissenborn, and Tim Salimans. Axial attention in multidimensional transformers. *arXiv preprint arXiv:1912.12180*, 2019.

Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, Quoc V. Le, and Adam Hartwig. Searching for mobilenetv3. 2019.

Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

Han Hu, Zheng Zhang, Zhenda Xie, and Stephen Lin. Local relation networks for image recognition. *arXiv preprint arXiv:1904.11491*, 2019.

Jie Hu, Li Shen, Samuel Albanie, Gang Sun, and Andrea Vedaldi. Gather-excite: Exploiting feature context in convolutional neural networks. In *Advances in Neural Information Processing Systems*, 2018a.

Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018b.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Learning Representations*, 2015.

Norman P. Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, Rick Boyle, Pierre-luc Cantin, Clifford Chao, Chris Clark, Jeremy Coriell, Mike Daley, Matt Dau, Jeffrey Dean, Ben Gelb, Tara Vazir Ghaemmaghami, Rajendra Gottipati, William Gulland, Robert Hagmann, C. Richard Ho, Doug Hogberg, John Hu, Robert Hundt, Dan Hurt, Julian Ibarz, Aaron Jaffey, Alek Jaworski, Alexander Kaplan, Harshit Khaitan, Daniel Killebrew, Andy Koch, Naveen Kumar, Steve Lacy, James Laudon, James Law, Diemthu Le, Chris Leary, Zhuyuan Liu, Kyle Lucke, Alan Lundin, Gordon MacKean, Adriana Maggiore, Maire Mahony, Kieran Miller, Rahul Nagarajan, Ravi Narayanaswami, Ray Ni, Kathy Nix, Thomas Norrie, Mark Omernick, Narayana Penukonda, Andy Phelps, Jonathan Ross, Matt Ross, Amir Salek, Emad Samadiani, Chris Severn, Gregory Sizikov, Matthew Snelham, Jed Souter, Dan Steinberg, Andy Swing, Mercedes Tan, Gregory Thorson, Bo Tian, Horia Toma, Erick Tuttle, Vijay Vasudevan, Richard Walter, Walter Wang, Eric Wilcox, and Doe Hyun Yoon. In-datacenter performance analysis of a tensor processing unit. *SIGARCH Comput. Archit. News*, 45(2):1–12, June 2017. ISSN 0163-5964. doi: 10.1145/3140659.3080246. URL `http://doi.acm.org/10.1145/3140659.3080246`.

Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. Transformers are rnns: Fast autoregressive transformers with linear attention. 2020.

Nikita Kitaev, Lukasz Kaiser, and Anselm Levskaya. Reformer: The efficient transformer. *arXiv preprint arXiv:2001.04451*, 2020.

Jungkyu Lee, Taeryun Won, Tae Kwan Lee, Hyemin Lee, Geonmo Gu, and Kiho Hong. Compounding the performance improvements of assembled techniques in a convolutional neural network, 2020.

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 *European Conference on Computer Vision*, pp. 740–755. Springer, 2014.

Ilya Loshchilov and Frank Hutter. SGDR: Stochastic gradient descent with warm restarts. In *International Conference on Learning Representations*, 2017.

John Nickolls and William J Dally. The gpu computing era. *IEEE micro*, 30(2):56–69, 2010.

Jongchan Park, Sanghyun Woo, Joon-Young Lee, and In So Kweon. Bam: bottleneck attention module. In *British Machine Vision Conference*, 2018.

Niki Parmar, Ashish Vaswani, Jakob Uszkoreit, Łukasz Kaiser, Noam Shazeer, Alexander Ku, and Dustin Tran. Image transformer. In *International Conference on Machine Learning*, 2018.

Ethan Perez, Florian Strub, Harm de Vries, Vincent Dumoulin, and Aaron C. Courville. Film: Visual reasoning with a general conditioning layer. *CoRR*, abs/1709.07871, 2017.

Prajit Ramachandran, Niki Parmar, Ashish Vaswani, Irwan Bello, Anselm Levskaya, and Jonathon Shlens. Stand-alone self-attention in vision models. *CoRR*, abs/1906.05909, 2019. URL `http://arxiv.org/abs/1906.05909`.

Tal Ridnik, Hussam Lawen, Asaf Noy, Emanuel Ben Baruch, Gilad Sharir, and Itamar Friedman. Tresnet: High performance gpu-dedicated architecture. 2020.

Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 4510–4520, 2018.

Peter Shaw, Jakob Uszkoreit, and Ashish Vaswani. Self-attention with relative position representations. *CoRR*, abs/1803.02155, 2018. URL `http://arxiv.org/abs/1803.02155`.

Noam Shazeer. Fast transformer decoding: One write-head is all you need. 2019.

Zhuoran Shen, Mingyuan Zhang, Shuai Yi, Junjie Yan, and Haiyu Zhao. Efficient attention: Self-attention with linear complexities. *CoRR*, abs/1812.01243, 2018. URL `http://arxiv.org/abs/1812.01243`.

Mingxing Tan and Quoc V. Le. Efficientnet: Rethinking model scaling for convolutional neural networks. *CoRR*, abs/1905.11946, 2019. URL `http://arxiv.org/abs/1905.11946`.

Yi Tay, Mostafa Dehghani, Dara Bahri, and Donald Metzler. Efficient transformers: A survey. 2020.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, pp. 5998–6008, 2017.

Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks. In *Advances in Neural Information Processing Systems*, 2015.

Huiyu Wang, Yukun Zhu, Bradley Green, Hartwig Adam, Alan Yuille, and Liang-Chieh Chen. Axial-deeplab: Stand-alone axial-attention for panoptic segmentation. 2020a.

Sinong Wang, Belinda Z. Li, Madian Khabsa, Han Fang, and Hao Ma. Linformer: Self-attention with linear complexity. 2020b.

Sanghyun Woo, Jongchan Park, Joon-Young Lee, and In So Kweon. Cbam: Convolutional block attention module. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 3–19, 2018.

Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhudinov, Rich Zemel, and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention. Proceedings of Machine Learning Research, pp. 2048–2057. PMLR, 2015.

Hang Zhang, Chongruo Wu, Zhongyue Zhang, Yi Zhu, Zhi Zhang, Haibin Lin, Yue Sun, Tong He, Jonas Mueller, R. Manmatha, Mu Li, and Alexander Smola. Resnest: Split-attention networks. 2020.

Hengshuang Zhao, Jiaya Jia, and Vladlen Koltun. Exploring self-attention for image recognition. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

Barret Zoph and Quoc V. Le. Neural architecture search with reinforcement learning. In *International Conference on Learning Representations*, 2017.

A GENERATING LAMBDAS FROM MASKED CONTEXTS

In some applications such as denoising tasks or auto-regressive training, it may be useful to restrict interactions to a sub-context $\mathcal{C}_n \subset \mathcal{C}$ when generating $\boldsymbol{\lambda}_n$ for query position n. For example, for *parallel* auto-regressive training, it is necessary to mask the future by ensuring that the output $\boldsymbol{y}_n$ only depends on past context positions $m < n$. Self-attention achieves this by zeroing out the irrelevant attention weights $\boldsymbol{a}_{nm'} = 0 \; \forall m' \notin \mathcal{C}_n$, thus guaranteeing that $\boldsymbol{y}_n = \sum_m \boldsymbol{a}_{nm} \boldsymbol{v}_m$ only depends on $\mathcal{C}_n$.

Similarly, we can block interactions between queries and masked context positions when generating lambdas by applying a mask before summing the contributions of context positions. Using the einsum notation, general masking can be implemented as

$$\begin{aligned}
\boldsymbol{\mu}^c_{bmkv} &= einsum(\boldsymbol{K}_{bmku}, \boldsymbol{V}_{bmvu}) \\
\boldsymbol{\lambda}^c_{bnkv} &= einsum(\boldsymbol{P}_{nm}, \boldsymbol{\mu}_{bmkv}) \\
\boldsymbol{\lambda}^p_{bnkv} &= einsum(\boldsymbol{E}_{knmu} * \boldsymbol{P}_{nm}, \boldsymbol{V}_{bmvu})
\end{aligned} \tag{5}$$

where $p_{nm} = 1[m \in \mathcal{C}_n]$ and $*$ is a broadcasted element-wise multiplication.

One can also normalize the keys by only considering the elements in their contexts. Computing *masked* lambdas still does not require to materialize per-example attention maps and the complexities are the same as for global lambdas case.

B Additional related work

In this section, we review the attention operation and related works on improving its scalability. We discuss connections between lambda layers and channel, spatial or linear attention mechanisms and show how they can be cast as *less flexible specific instances* of lambda layers. This section assumes $|u|$=1 to favor simplicity and highlight differences between lambda layers and attention mechanisms.

B.1 Softmax attention

Softmax attention Softmax-attention produces a distribution over the context for each query $\boldsymbol{q}_n$ as $\boldsymbol{a}_n = \text{softmax}(\boldsymbol{K}\boldsymbol{q}_n) \in \mathbb{R}^{|m|}$ where the keys $\boldsymbol{K}$ are obtained from the context $\boldsymbol{C}$. The attention distribution $\boldsymbol{a}_n$ is then used to form a linear combination of values $\boldsymbol{V}$ obtained from the context as $\boldsymbol{y}_n = \boldsymbol{V}^T\boldsymbol{a}_n = \sum_m a_{nm}\boldsymbol{v}_m \in \mathbb{R}^{|v|}$. As we take a weighted sum of the values, we transform the query $\boldsymbol{q}_n$ into the output $\boldsymbol{y}_n$ and discard its attention distribution $\boldsymbol{a}_n$. This operation captures content-based interactions, but not position-based interactions.

Relative attention In order to model position-based interactions, relative attention (Shaw et al., 2018) introduces a learned matrix of $|m|$ positional embeddings $\boldsymbol{E}_n \in \mathbb{R}^{|m|\times|k|}$ and computes the attention distribution as $\boldsymbol{a}_n = \text{softmax}((\boldsymbol{K} + \boldsymbol{E}_n)\boldsymbol{q}_n) \in \mathbb{R}^{|m|}$. The attention distribution now also depends on the query position n relative to positions of context elements m. Relative attention therefore captures both content-based and position-based interactions.

B.2 Attention with sparse patterns

A significant challenge in applying (relative) attention to large inputs comes from the *quadratic* $\Theta(|bnm|)$ memory footprint required to store attention maps. Many recent works therefore propose to impose specific patterns to the attention maps as a means to reduce the context size $|m|$ and therefore the memory footprint of the attention operation. These approaches include *local* attention patterns (Dai et al., 2019; Parmar et al., 2018; Ramachandran et al., 2019), *axial* attention patterns (Ho et al., 2019; Wang et al., 2020a), *static sparse* attention patterns (Child et al.; Beltagy et al., 2020) or *dynamic sparse* attention patterns (Kitaev et al., 2020). See Tay et al. (2020) for a review. Their implementations can be rather complex, sometimes require low-level kernel implementations to get computational benefits or may rely on specific assumptions on the shape of the inputs (e.g., axial attention).

In contrast, lambda layers are simple to implement for both global and local contexts using simple einsum and convolution primitives and capture *dense* content *and position-based* interactions with no assumptions on the input shape.

B.3 Linear attention

Another approach to reduce computational requirements of attention mechanisms consists in approximating the attention operation in linear space and time complexity, which is referred to as linear (or efficient) attention. Linear attention mechanisms date back to de Brébisson & Vincent (2016); Britz et al. (2017) and were later introduced in the visual domain by Chen et al. (2018); Shen et al. (2018). They are recently enjoying a resurgence of popularity with many works modifying the popular Transformer architecture for sequential processing applications (Katharopoulos et al., 2020; Wang et al., 2020b; Choromanski et al., 2020).

Linear attention via kernel factorization Linear attention is typically obtained by reinterpreting attention as a similarity kernel and leveraging a low-rank kernel factorization as

$$\text{Attn}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = \text{softmax}(\boldsymbol{Q}\boldsymbol{K}^T)\boldsymbol{V} \sim \phi(\boldsymbol{Q})(\phi(\boldsymbol{K}^T)\boldsymbol{V}) \tag{6}$$

for some feature function ϕ. Computing $\phi(\boldsymbol{K}^T)\boldsymbol{V} \in \mathbb{R}^{|k|\times|v|}$ first bypasses the need to materialize the attention maps $\phi(\boldsymbol{Q})\phi(\boldsymbol{K}^T)$ and the operation therefer has *linear* complexity with respect to the input length $|n|$.

Multiple choices for the feature function ϕ have been proposed. For example, Katharopoulos et al. (2020) use $\phi(\boldsymbol{x}) = \text{elu}(\boldsymbol{x}) + 1$, while Choromanski et al. (2020) use positive orthogonal random features to approximate the original softmax attention kernel. In the visual domain, both Chen et al.

(2018) and Shen et al. (2018) use $\phi(\boldsymbol{x}) = \text{softmax}(\boldsymbol{x})$. This choice is made to guarantee that the rows of the (non-materialized) attention maps $\phi(\boldsymbol{Q})\phi(\boldsymbol{K})^T$ sum to 1 as is the case in the regular attention operation.

B.4 Lambda layers vs linear attention

We now discuss the main differences between lambda layers and linear attention mechanisms.

1) Lambda layers extend linear attention to also consider position-based interactions. The kernel approximation from Equation 6 can be rewritten for a single query $\boldsymbol{q}_n$ as

$$\boldsymbol{y}_n = (\phi(\boldsymbol{K})^T\boldsymbol{V})^T\phi(\boldsymbol{q}_n) \tag{7}$$

which resembles the output of the *content lambda* $\boldsymbol{y}^c_n = (\boldsymbol{\lambda}^c)^T\boldsymbol{q}_n = (\bar{\boldsymbol{K}}^T\boldsymbol{V})^T\boldsymbol{q}_n$ from Equations 1 and 2. Lambda layers extend linear attention mechanisms to also consider position-based interactions as

$$\boldsymbol{y}_n = \boldsymbol{\lambda}_n^T\boldsymbol{q}_n = (\boldsymbol{\lambda}^c + \boldsymbol{\lambda}^p_n)^T\boldsymbol{q}_n = ((\bar{\boldsymbol{K}} + \boldsymbol{E}_n)^T\boldsymbol{V})^T\boldsymbol{q}_n \tag{8}$$

assuming $|u|$=1. In the above equation, computing the position (or content) lambda has $\Theta(|bmkv|)$ time complexity. As the position lambdas are not shared across query positions n, this cost is repeated for all $|n|$ queries, leading to a total time complexity $\Theta(|bnmkv|)$. Unlike linear attention mechanisms, lambda layers have *quadratic time complexity* with respect to the input length (in the global context case) because they consider position-based interactions.

2) Lambda layers do not necessarily attempt to approximate an attention kernel. While approximations of the attention kernel are theoretically motivated, we argue that they may be unnecessarily restrictive. For example, the kernel approximation in Equation 6 requires the *same* feature function ϕ on both $\boldsymbol{Q}$ and $\boldsymbol{K}$ and precludes the use of more flexible non-linearities and normalization schemes.

In contrast, lambda layers do not attempt to approximate an attention kernel. This simplifies their design and allows for more flexible non-linearity and normalization schemes, which we find useful in our ablations (See Table 12 in Appendix E). Considering the position embeddings independently of the keys notably enables a simple and efficient local implementation with the lambda convolution. Approximating the *relative* attention kernel would require normalizing the position embeddings with the keys (i.e., $\phi(\boldsymbol{K}+\boldsymbol{E}_n)$ instead of $\phi(\boldsymbol{K})+\boldsymbol{E}_n$), which cannot be implemented in the local context case with a convolution.

3) The lambda abstraction reveals the computational benefits of using a multiquery formulation. Finally, this work proposes to abstract the $\bar{\boldsymbol{K}}^T\boldsymbol{V}$ and $\boldsymbol{E}_n^T\boldsymbol{V}$ matrices as linear functions (the *content* and *position* lambdas) that are directly applied to the queries. The lambda abstraction reveals the benefits of multiquery lambdas (as opposed to the traditional multi-head attention formulation) as a means to reduce computational costs.

B.5 Channel and spatial attention.

Finally, we show that the lambda abstraction generalizes *channel* and *spatial* attention mechanisms, both of which can be viewed as specific instances of lambda layers. This observation is consistent with our experiments which demonstrate that lambda layers outperform both channel and spatial attention while being more computationally efficient.

Channel attention *Channel attention* mechanisms, such as Squeeze-and-Excitation (SE) and FiLM layers, recalibrate features via cross-channel interactions by aggregating signals from the entire feature map (Hu et al., 2018b;a; Perez et al., 2017). In particular, the SE operation can be written as $y_{nk} = w_k q_{nk}$ where w_k is the excitation weight for channel k in the query $\boldsymbol{q}_n$. This can be viewed as using a *diagonal* lambda which is *shared across query positions* $\boldsymbol{\lambda}_n = diag(w_1 \cdots w_{|k|})$. Channel attention mechanisms have proven useful to complement convolutions but cannot be used as a stand-alone layer as they discard spatial information.

Spatial attention Conversely, *spatial attention* mechanisms, reweigh each position based on signals aggregated from all channels (Xu et al., 2015; Park et al., 2018; Woo et al., 2018). These mechanisms can be written as $y_{nk} = w_n q_{nk}$ where w_n is the attention weight for position n in the input query $\boldsymbol{Q}$. This can be viewed as using (position-dependent) scalar lambdas $\boldsymbol{\lambda_n} = w_n \mathbb{I}$ where $\mathbb{I}$ is the identity matrix. Spatial attention has also proven helpful to complement convolutions but also cannot be used as a stand-alone layer as it discards channel information.

B.6 HYPERNETWORKS

LambdaNetworks can alternatively be viewed as an extension of HyperNetworks (Ha et al., 2016) that dynamically generate their computations based on the inputs contexts.

C Experimental details

ResNets. We use the ResNet-v1 implementation and initialize the γ parameter in the batch normalization (Ioffe & Szegedy, 2015) layer at the end of the bottleneck blocks to 0. Squeeze-and-Excitation layers employ a squeeze ratio of 4.

Lambda layer implementation details Unless specified otherwise, all lambda layers use query depth $|k|$=16, $|h|$=4 heads and intra-depth $|u|$=1. The *position* lambdas are generated with local contexts of size $|m|$=23x23 and the *content* lambdas with the global context as described in Equation 4. When the intra-depth is increased to $|u|$ >1, we reduce the scope of size $|m|$=7x7 and switch to the convolution implementation to reduce flops. The projections to compute $\boldsymbol{Q}$ and $\boldsymbol{V}$ are followed by batch normalization and $\boldsymbol{K}$ is normalized via a softmax operation. Positional embeddings are initialized at random using the unit normal distribution. Local positional lambdas can be implemented interchangeably with the lambda convolution or by using the *global* einsum implementation from Equation 4 and masking the position embeddings outside of the local contexts. The latter can be faster but has a higher memory footprint and FLOPS due to the $\Theta(knm)$ term (see Equation 4). In our experiments, we use the convolution implementation only for input length $|n| > 85^2$ or intra-depth $|u| > 1$.

LambdaResNets. We construct our LambdaResNets by replacing the spatial (3x3) convolutions in ResNet architectures by our proposed lambda layer, with the exception of the stem which is left unchanged. We apply 3x3 average-pooling with stride 2 after the lambda layers to downsample in place of the strided convolution. The number of residual blocks per stage for the deeper ResNets are [4, 29, 53, 4] for ResNet-270, [4, 36, 72, 4] for ResNet-350, and [4, 44, 87, 4] for ResNet-420. When working with hybrid LambdaNetworks, we use a single lambda layer in c4 for LambdaResNet50, 3 lambda layers for LambdaResNet101, 6 lambda layers for LambdaResNet-152/200/270/350 and 8 lambda layers for LambdaResNet-420. Lambda layers are uniformly spaced in the c4 stage for hybrid architectures.

ImageNet training setups. We consider two training setups for the ImageNet classification task. The 90 epochs training setup trains models for 90 epochs using standard preprocessing and allows for fair comparisons with classic works. The 350 epochs training setup trains models for 350 epochs using improved data augmentation and regularization and is closer to training methodologies used in modern works with state-of-the-art accuracies.

ImageNet 90 epochs training setup. We use the vanilla ResNet for fair comparison with prior works. We used the default hyperparameters as found in official implementations without doing additional tuning. All networks are trained end-to-end for 90 epochs via backpropagation using SGD with momentum 0.9. The batch size B is 4096 distributed across 32 TPUv3 cores (Jouppi et al., 2017) and the weight decay is set to 1e-4. The learning rate is scaled linearly from 0 to 0.1B/256 for 5 epochs and then decayed using the cosine schedule (Loshchilov & Hutter, 2017). We use batch normalization with decay 0.9999 and exponential moving average with weight 0.9999 over trainable parameters and a label smoothing of 0.1. The input image size is set to 224x224. We use standard training data augmentation (random crops and horizontal flip with 50% probability). Most papers compared against in Table 3 use a similar training setup and also replace the 3x3 spatial convolutions in ResNet architectures by their proposed methods. This allows for a fair comparison.

ImageNet 350 epochs training setup. Higher accuracies on ImageNet are commonly obtained by training longer with increased augmentation and regularization (Lee et al., 2020; Tan & Le, 2019). In the 350 epochs training setup, we replace the baseline architecture with the ResNet-D (He et al., 2018) and use squeeze-and-excitation in the residual blocks that do not employ lambda layers for the hybrid LambdaResNets. We additionally replace the max pooling layer in the stem by a strided 3x3 convolution. Networks are trained for 350 epochs with a batch size B of 4096 or 2048 distributed across 32 or 64 TPUv3 cores, depending on memory constraints. We employ RandAugment (Cubuk et al., 2019) with a magnitude of 15 as the data augmentation strategy. We use a smaller weight decay of 4e-5 and dropout with a drop probability of 0.3. All architectures deeper than ResNet-200 are trained with stochastic depth with a drop probability of 0.2.

Tuning Each training setup uses a constant set of hyperparameters across model scales. The improved 350 epoch training setup was found by tuning the baseline architectures to identify a robust training setup across different scales. While individual accuracies may be improved with further tuning, we favor simplicity and use the same training hyperparameters for all experiments. We do not perform early stopping and simply report the final accuracies.

Throughputs. Figure 2 reports the latency to process a batch of 4096 images on 32 TPUv3 cores using mixed precision training (ı.e bfloat16 activations). Table 4, Table 8 and Table 9 report inference throughput on 8 TPUv3 cores using float32 precision.

FLOPS count. We do not count zeroed out flops when computing positional lambdas with the einsum implementation from Equation 4. Flops count is highly dependent on the scope size which is rather large by default ($|m|$=23x23). In Table 11, we show that it is possible to significantly reduce the scope size and therefore FLOPS at a minimal degradation in performance.

Computational efficiency. In these experiments, we replace the last two stages of the ResNet architecture (where the convolutions are the most computationally expensive) with lambda layers. The parameter-efficient LambdaResNets in Table 5 employ an image size of 320. For flops efficiency, we additionally reduce the lambda scope size to $|m|$=7x7 and set the image size to 256.

COCO object detection. We employ the architecture from the improved ImageNet training setup as the backbone in the Mask-RCNN architecture. All models are trained on 1024x1024 images from scratch for 130k steps with a batch size of 256 distributed across 128 TPUv3 cores with synchronized batch normalization. We apply multi-scale jitter of [0.1, 2.0] during training. The learning rate is warmed up for 1000 steps from 0 to 0.32 and divided by 10 at steps 90, 95 and 97.5% of training. The weight decay is set to 4e-5.

D HYBRID LAMBDANETWORKS

In Table 8 and Table 9, we study the throughput and accuracy of hybrid LambdaNetwork architectures. We find that lambda layers are most helpful in the last two stages of the ResNet architecture (commonly referred to as *c4* and *c5*) when considering the speed-accuracy tradeoff (see Table 8). In particular, lambda layers in the c5 stage incur almost no speed decrease compared to 3x3 convolutions. Lambda layers in the c4 stage are relatively slower than convolutions but are crucial to reach high accuracies. In Table 9, we test how the speed and final accuracy is impacted by the number of lambda layers in the c4 stage. Our results reveal that most benefits from lambda layers can be obtained by 1) replacing a few 3x3 convolutions with lambda layers in the second last stage (commonly referred to as *c4*) of the ResNet architecture and 2) replacing all 3x3 convolutions in the last stage (*c5*). The resulting hybrid LambdaResNets architectures have increased representational power at a virtually negligible decrease in throughput compared to their vanilla ResNet counterparts.

Table 8: Inference throughput and top-1 accuracy as a function of lambda (L) vs convolution (C) layers' placement in a ResNet50 architecture on 224x224 inputs.

Architecture	Params (M)	Throughput	top-1
$\mathbf{C} \to \mathbf{C} \to \mathbf{C} \to \mathbf{C}$	25.6	7240ex/s	76.9
$\mathbf{L} \to \mathbf{C} \to \mathbf{C} \to \mathbf{C}$	25.5	1880ex/s	77.3
$\mathbf{L} \to \mathbf{L} \to \mathbf{C} \to \mathbf{C}$	25.0	1280ex/s	77.2
$\mathbf{L} \to \mathbf{L} \to \mathbf{L} \to \mathbf{C}$	21.7	1160ex/s	77.8
$\mathbf{L} \to \mathbf{L} \to \mathbf{L} \to \mathbf{L}$	15.0	1160ex/s	78.4
$\mathbf{C} \to \mathbf{L} \to \mathbf{L} \to \mathbf{L}$	15.1	2200ex/s	78.3
$\mathbf{C} \to \mathbf{C} \to \mathbf{L} \to \mathbf{L}$	15.4	4980ex/s	78.3
$\mathbf{C} \to \mathbf{C} \to \mathbf{C} \to \mathbf{L}$	18.8	7160ex/s	77.3

Table 9: Impact of number of lambda layers in the c4 stage of LambdaResNets. Most benefits from lambda layers can be obtained by having a few lambda layers in the c4 stage. Such hybrid approaches maximize the speed-accuracy tradeoff.

Config	Params (M)	Throughput	top-1
ResNet101 - 224x224			
Baseline	44.6	4600 ex/s	81.3
+ SE	63.6	4000 ex/s	81.8
+ 3 lambda	36.9	4040 ex/s	82.3
+ all lambdas	26.0	2560 ex/s	82.6
ResNet152 - 256x256			
Baseline	60.2	2780 ex/s	82.5
+ SE	86.6	2400 ex/s	83.0
+ 6 lambdas	51.4	2400 ex/s	83.4
+ all lambdas	35.1	1480 ex/s	83.4

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

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 11: Impact of the position lambda scope size on the ImageNet classification task. Flops significantly increase with scope size, however larger scopes do not translate to longer running time when using the einsum implementation (see Equation 4).

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

E ABLATIONS

Content vs position interactions Table 10 presents the relative importance of content-based and position-based interactions on the ImageNet classification task. As expected, position-based interactions are necessary to reach high accuracies, while content-based interactions only bring marginal improvements over position-based interactions.

Importance of scope size The small memory footprint of LambdaNetworks enables considering global contexts, even in the early high resolution layers of the networks. Table 11 presents flops counts and top-1 ImageNet accuracies when varying scope sizes for LambdaResNet50 on 224x224 inputs. We find benefits from using larger scopes, with a plateau around $|m|$=15x15, which validates the importance of long-range interactions. We choose $|m|$=23x23 as the default to account for experiments that use larger image sizes.

Normalization Table 12 ablates normalization operations in the design of the lambda layer. We find that normalizing the keys is crucial for performance and that other normalization functions besides the softmax can be considered. Additionally, applying batch normalization to the queries and values is also helpful.

Table 12: Impact of normalization schemes in the lambda layer.

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

Table 13: Ablations on the ImageNet classification task using the LambdaResNet50. All configurations outpeform the convolutional baseline at a lower parameter cost.

$\|k\|$	$\|h\|$	$\|u\|$	Params (M)	top-1
ResNet baseline			25.6	76.9
8	2	1	14.8	77.2
8	16	1	15.6	77.9
2	4	1	14.7	77.4
4	4	1	14.7	77.6
8	4	1	14.8	77.9
16	4	1	15.0	78.4
32	4	1	15.4	78.4
2	8	1	14.7	77.8
4	8	1	14.7	77.7
8	8	1	14.7	77.9
16	8	1	15.1	78.1
32	8	1	15.7	78.5
8	8	4	15.3	78.4
8	8	8	16.0	78.6
16	4	4	16.0	78.9

Varying query depth and number of heads. In Table 13, we study the impact of query depth $|k|$, number of heads $|h|$ and intra-depth $|u|$ on the accuracy. Our experiments indicate that LambdaNetworks outperform the convolutional and attentional baselines for a wide range of $|k|$ and $|h|$ hyperparameters. As expected, we get additional improvements by increasing the query depth $|k|$ or intra-depth $|u|$. The number of heads is best set at intermediate values such as $|h|$=4. A large number of heads $|h|$ excessively decreases the value depth $|v| = d/|h|$, while a small number of heads translates to too few queries, both of which hurt performance.

F Lambda layers in a resource constrained scenario

In this section, we study lambda layers in a resource-constrained scenario using the MobileNetv2 and MobileNetv3 architectures (Sandler et al., 2018; Howard et al., 2019). These architectures employ lightweight inverted bottleneck blocks which consist of the following sequence: 1) a pointwise convolution for expanding the number of channels, 2) a depthwise convolution for spatial mixing and 3) a final pointwise convolution for channel mixing. The use of a depthwise convolution (as opposed to a regular convolution) reduces parameters and flops, making inverted bottlenecks particularly well-suited for embedded applications.

Lightweight lambda layers. We obtain a lightweight lambda block as follows. We replace the depthwise convolution in the inverted bottleneck with a lambda layer. The lightweight lambda layer uses query depth $|k|$=32, $|h|$=4 and a *smaller scope size* $|m|$=5x5 to reduce computational demand. Finally, we change the first pointwise convolution to output the same number of channels (instead of increasing the number of channels) to reduce computations.

MobileNet experiments We wish to assess whether lambda layers can improve the flops-accuracy (or parameter-accuracy) tradeoff of mobilenet architectures. We therefore experiment with hybrid LambdaNetworks and replace only a few inverted bottlenecks so that the resulting architectures have similar computational demands as their baselines (see Table 14). For MobileNetv2, this simple strategy reduces parameters and flops by $\sim$10% while improving ImageNet accuracy by 0.6%. The improvement is smaller for the MobileNetv3 architecture. However we note that the MobileNetv3 architecture is a local optimum obtained via a combination of automated architecture search and human design assuming only access to inverted bottleneck blocks. We hypothesize that adding lambda layers in the search space might lead to stronger improvements.

Table 14: Lambda layers improve ImageNet accuracy in a resource-constrained scenario.

Architecture	Params (M)	FLOPS (M)	top-1
MobileNet-v2	3.50	603	72.7
LambdaMobileNet-v2	3.21	563	73.3
MobileNet-v3-large	5.48	438	75.1
LambdaMobileNet-v3-large	5.47	435	75.3

Mobilenet training setup. All mobilenet architectures are trained for 350 epochs on Imagenet with standard preprocessing at 224x224 resolution. We use the same hyperparameters as Howard et al. (2019). More specifically, we use RMSProp with 0.9 momentum and a batch size of 4096 split across 32 TPUv3 cores. The learning rate is warmed up linearly to 0.1 and then multiplied by 0.99 every 3 epochs. We use a weight decay 1e-5 and dropout with drop probability of 0.2