OmniNet: Omnidirectional Representations from Transformers

Yi Tay [* 1] **Mostafa Dehghani** [* 2] **Vamsi Aribandi** [1 3] **Jai Gupta** [1] **Philip Pham** [1]
Zhen Qin [1] **Dara Bahri** [1] **Da-Cheng Juan** [1] **Donald Metzler** [1]

Abstract

This paper proposes Omnidirectional Representations from Transformers (OMNINET). In OmniNet, instead of maintaining a strictly horizontal receptive field, each token is allowed to attend to all tokens in the entire network. This process can also be interpreted as a form of extreme or intensive attention mechanism that has the receptive field of the entire width and depth of the network. To this end, the omnidirectional attention is learned via a meta-learner, which is essentially another self-attention based model. In order to mitigate the computationally expensive costs of full receptive field attention, we leverage efficient self-attention models such as kernel-based (Choromanski et al., 2020), low-rank attention (Wang et al., 2020) and/or Big Bird (Zaheer et al., 2020) as the meta-learner. Extensive experiments are conducted on autoregressive language modeling (LM1B, C4), Machine Translation, Long Range Arena (LRA), and Image Recognition. The experiments show that OmniNet achieves considerable improvements across these tasks, including achieving state-of-the-art performance on LM1B, WMT'14 En-De/En-Fr, and Long Range Arena. Moreover, using omnidirectional representation in Vision Transformers leads to significant improvements on image recognition tasks on both few-shot learning and fine-tuning setups.

1. Introduction

Transformers (Vaswani et al., 2017), characterized by stacked self-attention modules and feed-forward transformations, have become a staple in modern deep learning, natural language processing (Devlin et al., 2018; Raffel et al., 2019) and even computer vision (Dosovitskiy et al., 2020). One

key defining characteristics in the self-attention mechanism is the global receptive field in which each token is accessible to every other token in the sequence, serving as an enabler for learning global contextual representations.

This paper proposes learning omnidirectional representations from transformers. The key idea is to move beyond horizontally global receptive fields and explore the possibility of omnidirectional receptive fields. In short, we allow each token to not only attend to all other tokens in the same layer, but also all token in all the layers of the network. This global access enables tokens to have a full view of the network and as a result access the knowledge and intermediate representations of every token at each layer. By modeling the relationships amongst tokens of different hierarchical levels, we are also able to capture patterns pertaining to the propagation of representations across time. Finally, this approach can be also be interpreted as a form of dense residual connection (Huang et al., 2017), which has been shown to be beneficial by aiding gradient flow.

Learning omnidirectional receptive fields is non-trivial for two key reasons. Firstly, given the quadratic complexity of the scaled dot product attention, the complexity of designing such a receptive field is increased from $N^2 L$ to $(NL)^2$, where L is the depth of the network and N is the sequence length. We postulate that this is one big challenge that has prohibited this type of architecture from being explored in the past. Secondly, simply enabling omnidirectional attention from the get-go would easily cause a degeneration of the transformer into a flat network, losing much of its representational power that is enabled by sequentially refining its representations across network layers.

To mitigate these issues, our omnidirectional attention is implemented as a form of meta-learner that acts upon a standard transformer model. The meta-learner is yet another self-attention model that accepts all hidden representations across all layers as an input and refines them based on all the available information. In order to mitigate the prohibitive memory and computational costs of omnidirectional attention, we explore and evaluate multiple efficient alternatives of parameterizing the meta-learner, e.g., including fast attention via generalizable kernel attention (Choromanski et al., 2020), low-rank self-attention (Wang et al., 2020),

---
[*]Equal contribution [1]Google Research, Mountain View [2]Google Brain Team, Amsterdam [3]Google AI Resident. Correspondence to: Yi Tay <yitay@google.com>, Mostafa Dehghani <dehghani@google.com>.

and/or block-based sparsity (Zaheer et al., 2020). Additionally, we further hypothesize that employing methods that try to learn the low-rank factorized structure of the entire network can lead to improved generalization capabilities - as demonstrated in our few-shot learning experiments.

Aside from varying the parameterization of the meta-learner, we also introduce partitioned variants of OmniNet in which the meta-learner is applied to blocks of p consecutive layers. In short, this partitioning strategy groups the full network of L layers into $\frac{L}{p}$ partitions. After computing each partition, the meta-learner learns the omnidirectional attention of all nodes across all layers in the partition.

Via extensive experiments, we show that OmniNet achieves very promising results on a myriad of language, vision, and logic tasks. Specifically, we report strong experimental results on autoregressive language modeling (Chelba et al., 2013; Raffel et al., 2019), five collections of WMT machine translation, Long Range Arena (Tay et al., 2020a), and Image Recognition using Vision Transformers (Dosovitskiy et al., 2020). Moreover, we systematically evaluate OmniNets through the lens of the performance-compute trade-off and show that they are pareto-optimal in this regard.

On machine translation, OmniNet outperforms ADMIN (Liu et al., 2020), the current state-of-the-art 60 layers deep transformer model on two well-established machine translation collections (WMT'14 English-German and WMT'14 English-French). On the one billion language modeling benchmark, OmniNet outperforms existing state-of-the-art models such as Transformer-XL (Dai et al., 2019). On LRA, OmniNet improves aggregate performance over Performers (Choromanski et al., 2020) by $+8.9\%$ and vanilla Transformers by $+2.6\%$. On Image Recognition tasks, OmniNet demonstrates stellar few-shot learning and finetuning performance, outperforming ViT (Dosovitskiy et al., 2020) by up to $\approx +3\%$ on both finetuning and few-shot learning experiments.

2. Related Work

Just across the past several years, attention mechanisms (Bahdanau et al., 2014) have made a significant impact on machine learning research (Vaswani et al., 2017; Devlin et al., 2018; Dosovitskiy et al., 2020; Raffel et al., 2019; Brown et al., 2020; Dehghani et al., 2018). Simply speaking, these parameterized pooling mechanisms learn to align representations and route information based on the notion of relative importance. While early work in this area was mainly concerned with learning an alignment function between two or more sequences (Bahdanau et al., 2014; Parikh et al., 2016), there have been more focus on self-alignment (e.g., self-attention) in the recent research climate (Vaswani et al., 2017).

Attention mechanisms are generally applied layer-wise and

operate across a one-dimensional sequence. Attention is generally bidirectional, or unidirectional in the case where a token is to be denied access to future tokens. There have been early attempts to mix information across layers in pursuit of improving gradient flow and model trainability. For example, (Bapna et al., 2018) proposed transparent attention in which the decoder gains access to all encoder layers. (He et al., 2018) proposed layer-wise coordination between encoder-decoder for machine translation. (Tay et al., 2018) proposed to densely connect the attention across stacked RNN encoder layers for language understanding tasks. The recent Realformer (residual attention) (He et al., 2020) proposed connecting the attention activations in a residual fashion. We believe there is sufficient evidence in the literature to suggest that mixing representations across layers is beneficial. This is further supported by fundamental work in this area such as ResNets (He et al., 2016), highway networks (Srivastava et al., 2015) and DenseNets (Huang et al., 2017).

In this paper, we are mainly interested in methods for efficiently learning omnidirectional attention - an attention over the entire width and depth of the network. To this end, we leverage the recent advances in making transformers fast and efficient (Zaheer et al., 2020; Choromanski et al., 2020; Wang et al., 2020). Many of these approaches learn an approximation via low-rank projection, kernels or block-based sparsity. An overview and extensive empirical comparison can be found at (Tay et al., 2020b;a). To this end, the proposed approach leverages these recent advances to make what was previously not possible. By leveraging fast and efficient self-attention, we enable scalable and powerful omnidirectional attention.

3. The Proposed Method

This section introduces OmniNet. We first begin by reviewing the standard Transformer architecture.

3.1. Transformer Architectures

This section provides a brief background for the Transformer architecture. The Transformer block accepts $N \times d$ input, where N denotes the number of tokens in the sequence and d denotes the size of the representation. Each Transformer block is characterized by a self-attention block and a two layered feed-forward network with ReLU activations in-between that is applied position-wise.

3.1.1. SELF-ATTENTION

The self-attention mechanism first projects each input X into Q, K, V representations using linear transformations, corresponding to queries, keys, and values. The self-attention mechanism is typically multi-headed where multiple similar linear projections are executed in parallel. The output of
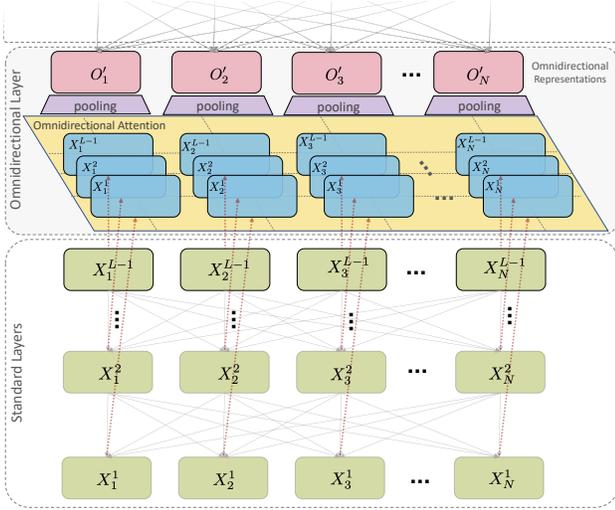
Figure 1. Overview of OmniNet. In the diagram, the omnidirectional module, when partition size is $P = L$, combines the information across all positions $(1 : N)$, across all layers $(1 : L-1)$, and for each position selects the best of all layers via a pooling operation to generate the final representations.

each self-attention head h at layer l is written as:

$$y_{h,l} = \text{softmax}\left(\frac{Q_{h,l}K_{h,l}^\top}{\sqrt{d_k}}\right)V_{h,l}, \qquad (1)$$

where $y_{h,l}$ is the output of head h at layer l and d_k is the size of each head. The output from the multiple heads is then concatenated and then passed through another linear transformation via $W_{o,l}$ which projects the concatenation of all heads down to d_m. This is wrapped via a layer normalization followed by a residual connection and can be written as: $\text{LayerNorm}(W_{o,l}\text{concat}([y_{1,l}\cdots y_{H,l}])) + x_{l-1}$ as the final output of the self-attention module.

Feed Forward Layers The FFN block of the Transformer block performs a two layer transformation defined as:

$$z_l = \text{LayerNorm}(W_{1,l}\text{ReLU}(W_{2,l}(Y))) + z_{l-1}, \qquad (2)$$

where W_1, W_2 are trainable parameters (weight transforms) of the FFN layer. Bias parameters are omitted for clarity.

3.2. OmniNet

The proposed OmniNet method operates on an arbitrary multi-layered architecture that accepts sequential inputs. In our description, this typically refers to a stacked X-former architecture in this section. Note that while this is typically a transformer model, it can also be an arbitrary variant (Choromanski et al., 2020; Wang et al., 2020). Figure 1 illustrates a brief overview of the proposed OmniNet architecture.

3.2.1. OMNIDIRECTIONAL REPRESENTATIONS

In a stacked network of L layers, each layer exposes a sequence of N vectors of d dimensions each. Specifically, OmniNet operates across all layers and connects the multi-layered network architecture in a **grid** like fashion. We describe the network as xformer which accepts X as an input and returns a tensor of $L \times N \times d$ dimensions.

$$\text{xformer}(X) = X_1, X_2 \cdots X_L, \qquad (3)$$

where $X_i \in \mathbb{R}^{N \times d}$. Let X_j^i be the representation of X at layer i and position j of the sequence. The OmniNet mechanism can be written as:

$$O = \text{Attend}(\text{IndexSort}(X_1, X_2, \cdots X_L)), \qquad (4)$$

where Attend denotes an arbitrary self-attention block. The IndexSort operation takes X_1, X_2, X_L and sorts,[1] tokens within each matrix by index such that the adjacent token of the ith token in layer l are the ith token from $l-1$ and $l+1$ respectively. Next, given that the input sequence length is LN, it is advantageous for Attend to be as efficient as possible. We describe three variants of OmniNet's core linear-time self-attention mechanism in subsequent sections.

Given $O \in \mathbb{R}^{(L \times N) \times d}$, the output of the omnidirectional attention, we perform $P(.)$ a pooling operator. While there are many choices of pooling operators, parameterized or otherwise, we adopt a simple pooling function - a max pooling of stride L.

$$O' = \text{MaxPool1D}(O), \qquad (5)$$

where $O' \in \mathbb{R}^{N \times d}$. Given O', the final representation of an OmniNet augmented network is defined as:

$$\text{OmniNet}(X) = \text{xformer}(X)_L + O'. \qquad (6)$$

The OmniNet and main transformer model are trained together in an end-to-end fashion, i.e., gradients flow to both networks concurrently at each backward pass.

3.2.2. MAINTAINING CAUSALITY AND AUTOREGRESSIVE DECODING

A key point to note with IndexSort is that this order enables us to apply a causal mask to the Attend function, namely if tokens are sorted according to sequence index first as opposed to layer first, then it would be easy to apply a causal mask M, where $M[i,j] = 0$ when $i \leq j$ and $-\inf$ when $i > j$. This enables OmniNet to be used in autoregressive settings.

---

[1] Since attention is permutation invariant this sorting simply makes it easier to (1) compute casual masks and (2) aggregate representations index-wise.

3.2.3. EFFICIENT TRANSFORMERS

We describe several choices of linear-time self-attention mechanisms that are used in OmniNet's omnidirectional attention. Generally, Attend refers to an attention block with an attention function and a two-layered positional FFN in a similar structure to the transformer backbone. For the sake of brevity, we only describe the core attention mechanism here. Our choice of the efficient transformer is informed by (Tay et al., 2020a) selecting models that perform well on the compute-performance trade-off. For a list of potential variants to adopt, we refer readers to (Tay et al., 2020b).

Kernel-based This variant uses the generalizable kernel attention, the fast attention mechanism proposed in Performer (Choromanski et al., 2020). Specifically, this is written as:

$$o = W_o \text{concat}(\hat{D_h}^{-1}(\phi(Q_h)(\phi(K_h))^\top V_h)),$$

where $\hat{D_h} = \text{diag}\phi(Q_h)((\phi(K_h))^\top 1_L)$ and $\phi(.)$ is a random feature map that projects $\mathbb{R}^d$ to $\mathbb{R}^r$. We refer readers to (Choromanski et al., 2020) for more details.

Low-rank Inspired by Linformer's (Wang et al., 2020) self-attention mechanism, we set Attend to be:

$$o = W_o(\text{concat}(\text{softmax}\left(\frac{Q_h(WK_h)^\top}{\sqrt{d_k}}\right)(WV_h)),$$

where $W \in \mathbb{R}^{N \times k}$ are low-rank projection transformations that are shared across heads and across keys and values. The complexity of this self-attention mechanism is Nk instead of N^2, where $k \ll N$.

Block and Memory based Lastly, we also explore a block and memory-based variant of efficient Transformers based on Big Bird (Zaheer et al., 2020). In short, this is a combination of windowed attention, global attention, and sparse attention. The output for token i is defined as:

$$o_i = x_i + \sum_{h=1}^{H} \text{softmax}\left(Q_{h,i}K_{h,N(i)}^\top\right)V_{h,i},$$

where $N(i)$ is the neighborhood function which denotes the out-neighbors of node i, H is the total number of heads and h represents a head. The neighborhood function is mainly dependent on the width of the windowed attention. We refer the reader to (Zaheer et al., 2020) for more details.

3.2.4. PARTITIONED OMNINET

This section describes the types of partitioning variants that we explore in OmniNet. When L is large, the eventual representation input to OmniNet can be extremely large.[2]

---

[2]A sequence length of $1K$ would result in a $11K$ input sequence length for a 12 layered Transformer model, when using an omnidirectional layer as the final layer.

Table 1. Experimental results (quality, i.e., perplexity scores at 30K and 100K respectively) on autoregressive language modeling. All models are approximately 50M parameters.

Model	LM1B	C4
Transformer	33.14	34.86
Realformer	32.95	35.63
Performer	34.33	35.68
BigBird	32.90	38.36
OmniNet$_B$	33.69 (-1.7%)	34.73 (+0.4%)
OmniNet$_P$	30.19 (+9.0%)	33.97 (+2.6%)
OmniNet$_T$	**30.12** (+9.1%)	**33.39** (+4.2%)

Table 2. Comparison with existing state-of-the-art and published works on One Billion Word Language modeling (Chelba et al., 2013) benchmark.

Model	#Params	PPL
Adaptive Input (Baevski & Auli)	0.5B	24.1
Adaptive Input (Baevski & Auli)	1.0B	23.7
Transformer-XL (Dai et al.)	0.5B	23.5
Transformer-XL (Dai et al.)	0.8B	21.8
OmniNet$_P$ (Large)	0.1B	21.6
OmniNet$_B$ (Large)	0.1B	22.0
OmniNet$_T$ (Large)	0.1B	**21.5**

Let P be an integer valued hyperparameter that determines the partition size. For a L layer transformer network, when $\ell \mod P$ is 0, we insert a meta-learner block.

$$X_\ell = \begin{cases} \text{Attend}(X_{\ell-P}, \cdots X_{\ell-1})), & \text{if } \ell \mod P = 0 \\ \text{xformer}(X_{\ell-1}) \end{cases}$$

In short, whenever $\ell \mod P = 0$, we activate an omnidirectional attention layer, aggregating representations all the way from the previous partition $\ell - P$ layer up till $\ell - 1$. In this case, we skip the original xformer layer, hence **maintaining approximately the same parameter size** of the network.

4. Experiments

We conduct experiments on autoregressive language modeling, machine translation, long range sequence modeling and a series of image recognition tasks. Our implementation uses Flax (Heek et al., 2020) and Jax (Bradbury et al., 2018).

4.1. Autoregressive Language Modeling

We run experiments on large-scale autoregressive (unidirectional) language modeling. We use two **large-scale** datasets, language modeling one billion (LM1B) (Chelba et al., 2013) and the Colossal Cleaned Common Crawl corpus (C4) (Raffel et al., 2019).

Table 3. Results on five collections from the WMT'17 machine translation task.

Model	En-De	En-Fi	Cs-En	En-Fr	Ru-En
Transformer.	28.6	20.5	22.2	43.0	35.8
OmniNet$_L$	28.8 (+0.7%)	20.8 (+1.5%)	22.8 (+2.7%)	**43.3** (+0.7%)	36.2 (+1.1%)
OmniNet$_B$	28.8 (+0.7%)	20.9 (+2.0%)	22.6 (+1.8%)	43.2 (+0.5%)	34.2 (-4.5%)
OmniNet$_P$	**29.0** (+1.4%)	20.9 (+2.0%)	**23.0** (+3.6%)	43.1 (+0.2%)	36.2 (+1.1%)

Table 4. Comparisons with the state-of-the-art on WMT'14 En-De and WMT'14 En-Fr. OmniNet outperforms ADMIN (Liu et al., 2020), the current state-of-the-art deep transformer model for MT.

Model	En-De	En-Fr
Evolved Trans. (So et al., 2019)	29.2	n/a
Large Trans. (Ott et al., 2018)	28.6	41.4
60L Trans. (Liu et al., 2020)	29.5	41.8
OmniNet$_P$	**29.8**	**42.6**

Table 5. Results on Long Range Arena (Tay et al., 2020a).

Model	Text	Retrieval	ListOps	Avg
Linformer	53.9	52.3	35.7	47.3
BigBird	64.0	54.7	36.1	51.6
Performer	65.4	53.8	18.0	45.7
+OmniNet$_P$	**65.6**	60.9	18.2	48.2
+OmniNet$_L$	63.1	**63.7**	**37.1**	54.6
Transformer	62.5	57.5	36.4	52.1
+OmniNet$_P$	**65.1**	58.8	**37.2**	53.7
+OmniNet$_L$	63.1	**63.8**	37.2	**54.7**

4.1.1. EXPERIMENTAL SETUP

For both tasks, we use a max length of 256 subword tokens per example and report scores on subword perplexity on the validation set. In the first ablative experiment, we train all models for $30K$ for LM1b and $100K$ steps for C4 using 16 TPU-V3 Chips. Models are of *base* size and have an embedding dimension of 512, 8 heads, 6 layers and hidden dimensions (MLP) of 2048. For strong baselines, we compare with Transformers, Performers (Choromanski et al., 2020), and BigBird (Zaheer et al., 2020). We also add the recent Realformer (residual attention Transformer) (He et al., 2020) as a strong baseline. For OmniNet, we tune the partition amongst {2,3,6}. All models have approximately $50M$ parameters. Next, we are interested in (1) how OmniNet scales to large sizes and (2) comparing with other published works (Dai et al., 2019). Hence, we implement a larger OmniNet with MLPs of size 8K and head size of 2K.

4.1.2. RESULTS ON LANGUAGE MODELING

Table 1 reports results on LM. We observe that OmniNet$_{P,T}$ outperforms all baselines by about $+9.1\%$ on LM1b and $+4.2\%$ on C4. We also outperform strong baselines such as Realformer, BigBird, and vanilla Transformers on both corpora. We also observe that OmniNet$_P$ performs reasonably close to OmniNet$_T$, which ascertains that using an efficient Transformer may be sufficient for omnidirectional attention. On the other hand, Table 2 reports a comparison with other published works on LM1B. Notably, OmniNet$_{P,T}$ (large) outperforms Transformer-XL and achieves state-of-the-art performance.

4.2. Neural Machine Translation

We conduct experiments on machine translation, a sequence-to-sequence task. for evaluating Transformer models.

4.2.1. EXPERIMENTAL SETUP

We use five collections/datasets from WMT-17,[3] namely En-De (English → German), En-Cs (English → Czech), En-Fi (English → Finnish), En-Fr (English → French) and En-Ru (English → Russian). We train all models using 16 TPU-V3 chips with a batch size of 256. Our base Transformer model has 6 layers, a hidden size of 4096, embedding size of 1024, and a head size of 1024. The number of heads is 16. We use a SentencePiece (Kudo & Richardson, 2018) vocabulary of $32K$ built for each language specifically. More details can be found in the appendix.

4.2.2. RESULTS ON WMT'17 MACHINE TRANSLATION

Table 3 reports results on all 5 collections of WMT'17. Overall, OmniNet$_P$ outperforms the vanilla Transformer model on all five collections, with up to $\approx +3.6\%$ performance improvement. Similar to LM, we find that the performer variant works the best and the BigBird variant works the worse.

4.2.3. COMPARISONS AGAINST STATE-OF-THE-ART

We train a large OmniNet model and compare it with the state-of-the-art approaches. We compare with ADMIN (Liu et al., 2020), a very deep (60 layers) Transformer model that achieves state-of-the-art performance on the WMT En-De dataset. We use a 8 layer OmniNet model with 4096 MLP dimensions, 1024 hidden dimensions and embedding dimensions. We compare models using *sacrebleu* (Post, 2018). For OmniNet, given the strong performance of the Performer variant on WMT'17 collections, we only train a single P variant OmniNet for comparing with SOTA models. Further details of the setup can be found in the appendix.

---

[3]http://www.statmt.org/wmt17/translation-task.html

Table 4 reports results on WMT'14 En-De and En-Fr. Our results show that OmniNet outperforms the existing state-of-the-art ADMIN model (Liu et al., 2020), a 60-layer deep transformer model.

4.3. Long Range Arena

We conduct experiments on the recently proposed Long Range Arena benchmark (Tay et al., 2020a). The goal of this experiment is to show that OmniNet improves long-range sequence modeling. A dual goal is to show that it is possible to combine different inductive biases to obtain a better efficient Transformer model that is versatile on different types of data.

4.3.1. EXPERIMENTAL SETUP

We run two key experiments using Transformer and Performer as the main backbone model and vary the meta-learner in OmniNet, using Linformer and Performer variants of the OmniNet meta-learner. The goal is to demonstrate that OmniNet translates to backbone agnostic improvements. We run OmniNet experiments using the LRA codebase and run OmniNet models using the same hyperparameters as the results reported in (Tay et al., 2020a). Note that LRA is comprised of five benchmarks, however, we omit Image and Pathfinder experiments since the best hyperparameters on these tasks turn out to be shallow (1-2 layered) models. We report the average of the text, retrieval, and ListOps tasks.

4.3.2. RESULTS ON LRA

Table 5 reports the results on our LRA experiments. Firstly, we observe that OmniNet makes substantial improvements to the base model, regardless of whether it is a Transformer or Performer. Notably, with OmniNet$_L$, the Linformer meta-learner, the Performer model is improved by almost 6 to 7 absolute percentage points. An interesting observation can be made on the ListOps task where Omninet$_P$ (Performer variant) does not result in much improvement over the base Performer. However, the performance doubles with OmniNet$_L$. Since the base Linformer model does pretty well on this task, we postulate that this is due to OmniNet$_L$ providing a Linformer-like inductive bias to the Performer model. Finally, we observe that OmniNet improves the vanilla Transformer in both cases (P or L), improving the average score by about $+2.6\%$ absolute percentage points.

4.4. Image Recognition

Transformer-based models started showing competitive performance on different vision tasks like classification, object detection, and segmentation (Chen et al., 2020; Dosovitskiy et al., 2020; Carion et al., 2020; Kumar et al., 2021).

To showcase the power of omnidirectional representations in yet another task, we incorporate the omnidirectional representation in Vision Transformer (ViT) (Dosovitskiy et al., 2020), when pre-trained on a large amount of data in a supervised fashion and evaluated on downstream image recognition tasks, either through few-shot learning or fine-tuning.

4.4.1. VISION TRANSFORMER

Vision Transformers (ViT) (Dosovitskiy et al., 2020) have recently shown impressive results on image classification compared to state-of-the-art convolutional networks, while they require significantly fewer computational resources to train. ViT is a standard Transformer that is directly applied to images. To do so, we first split the input images into non-overlapping patches and embedded them using a linear projection. The patch embeddings are provided as a sequence of tokens to a Transformer. When pre-trained on large datasets (14M-300M images) at a sufficient scale, ViT shows excellent results that are transferable to tasks with fewer data points.

4.4.2. EXPERIMENTAL SETUP

Similar to the ViT setup, we pre-train our OmniNet models on the JFT dataset (Sun et al., 2017) with 18k classes and 303M images, for 7 epochs. We evaluate our models in the transfer setup (few-shot and fine-tuning) on several downstream tasks: ImageNet, CIFAR-10, CIFAR-100 (Krizhevsky et al., 2009), Oxford-IIIT Pets (Parkhi et al., 2012), and Oxford Flowers-102 (Nilsback & Zisserman, 2008). We follow the pre-processing from (Kolesnikov et al., 2019) on both upstream and downstream datasets, which is used in the original ViT experiments.

In our experiments, we train and evaluate OmniNet$_{B/32}$ and OmniNet$_{B/16}$, which are based on ViT$_{B/32}$ and ViT$_{B/16}$.[4] Similar to ViT$_{B/32}$ and ViT$_{B/16}$, OmniNet$_{B/32}$ and OmniNet$_{B/16}$ are both "*base*" models, adopted from BERT, and use patch sizes of 32×32 and 16×16 respectively.

In our OmniNet models, we replace the final layer of ViT with an omnidirectional layer. In other words, we set the portion size $P = 12$. In this task, we limit our experiments to using Performers (Choromanski et al., 2020) in the omnidirectional attention, given their strong results among the efficient transformer variants.

During pre-training, we use a batch size of 4096 using Adam with $\beta_1 = 0.9$ and $\beta_2 = 0.999$, and use a weight decay of 0.05 for OmniNet. We use a learning rate of $8e{-}4$ with a linear decay and a linear warmup of $10K$ steps.

---

[4]Note that SOTA results on the downstream tasks we use here are from ViT$_{H/14}$ (Dosovitskiy et al., 2020), which has more than seven times as many parameters as the *base* models we use as baselines. Here, we aim at merely showcasing the gain of using omnidirectional representations in the image recognition task.

Table 6. Transfer performance of pre-trained OmniNet and equivalent ViT models in fine-tuning setup on popular image classification benchmarks. All models are pre-trained on the JFT-300M dataset and fine-tuned on the target dataset.

	$\text{ViT}_{B/32}$	$\text{OmniNet}_{B/32}$	$\text{ViT}_{B/16}$	$\text{OmniNet}_{B/16}$
ImageNet	$0.8073 \rightarrow$	0.8374	$0.8415 \rightarrow$	0.8626
CIFAR-10	$0.9861 \rightarrow$	0.9900	$0.9900 \rightarrow$	0.9940
CIFAR-100	$0.9049 \rightarrow$	0.9153	$0.9186 \rightarrow$	0.9224
Oxford-IIIT Pets	$0.9340 \rightarrow$	0.9441	$0.9580 \rightarrow$	0.9674
Oxford Flowers-102	$0.9927 \rightarrow$	0.9954	$0.9956 \rightarrow$	0.9961
exaFLOPs	165	193	743	891
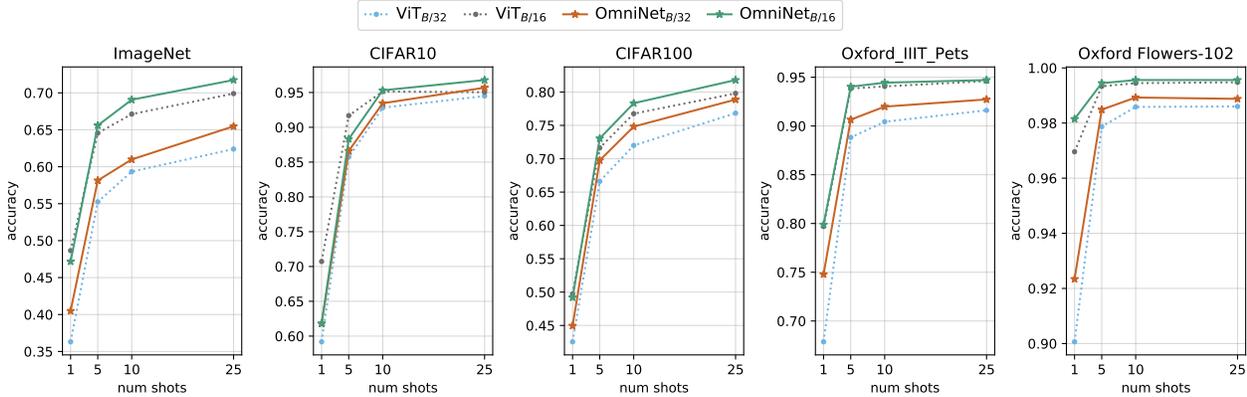
Figure 2. Performance of pre-trained OmniNet and equivalent ViT models in few-shot learning setup on downstream tasks, when transferred using only few images (1, 5, 10, and 25) per class.

4.4.3. RESULTS ON IMAGE RECOGNITION

We first present the results of OmniNet and corresponding ViT models as baselines in the fine-tuning setup. For fine-tuning, we use SGD with momentum and a batch size 512 in all downstream tasks. Table 6 presents the results of fine-tuning experiments. We also report the total pre-training compute, in terms of number of FLOPs for each model. As we can see, introducing a module that learns omnidirectional representations to Vision Transformers leads to improvements on different downstream tasks. Given these improvements and comparing the number of FLOPs for OmniNets and ViTs, we can see that the additional compute, thanks to efficient attention techniques, is fairly reasonable.

For evaluating OmniNet in the few-shot learning setup, similar to ViT, we train a linear classifier on top of the representations from the frozen pre-trained model, given only a subset of training examples. Plots in Figure 2 illustrate the accuracy of OmniNet and ViT, using different numbers of shots. The results indicate that adding omnidirectional representations to ViT leads to better transfer across all downstream datasets.

4.5. Effect of Partition Size and Compute/Performance Trade-offs

OmniNet offers the flexibility to apply the Omnidirectional layers on different partition sizes. With smaller partition sizes, we attend to tokens from fewer layers, and with
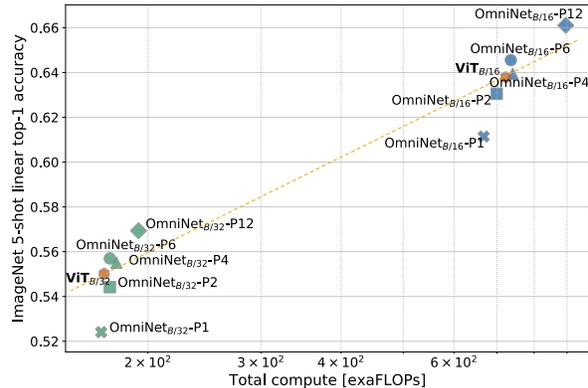

Figure 3. Performance of ViT and OmniNet (with different partition sizes) in terms of top-1 accuracy on ImageNet 5-shot linear, versus their computational costs in terms of number of FLOPs.

bigger partition, we widen the vertical receptive field of the omnidirectional attention, which might be effective for learning better representations by capturing information from more levels. In terms of computational costs, however, there is a trade-off when choosing the partition size. Small partition sizes means running attention on smaller sequences while repeating it more frequent, and bigger partition sizes means dealing with longer sequences, but having fewer omnidirectional layers in the network.

We train $\text{OmniNet}_{B/32}$ and $\text{OmniNet}_{B/16}$ with different partition sizes: $P = \{1,2,4,6,12\}$. Partition size $P = 1$ is simply having no vertical attention and it is just replacing normal attention in ViT, with Performer. We compare these
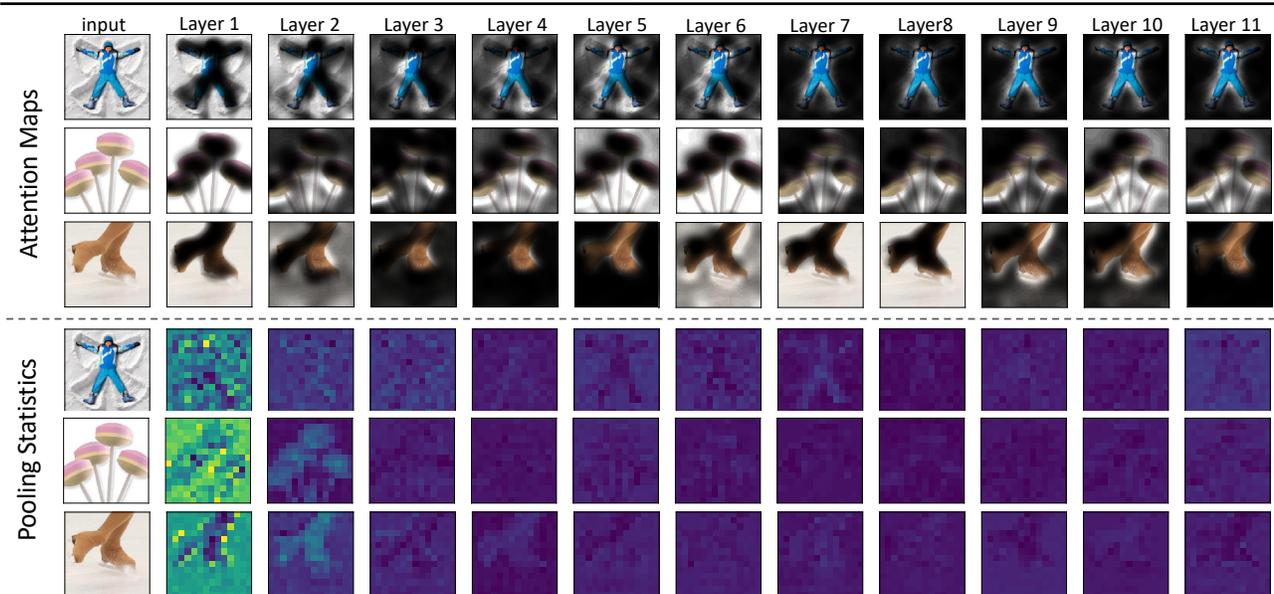
Figure 4. Contribution of different layers in Omnidirectional representations for a given set of examples. On top, we plot the omnidirectional attention maps (using OmniNet$_{B/16}$-P12) of one of the heads, over all layers, when CLS token in the last layer is used as query. On the bottom, we show the contribution of each layer to the pooling operation of the Omnidirectional module.

models in terms of their linear 5-shot accuracy on ImageNet dataset (similar to the ablation studies in (Dosovitskiy et al., 2020)). Figure 3 presents the performance of each model as well as their computational cost during pre-training.

A few patterns can be observed. For both OmniNet$_{B/32}$ and OmniNet$_{B/16}$, the power of omnidirectional directional representations kicks in when we work with partition sizes of more than 2. The input resolution during pre-training is 224×224, so for /32 and /16 models the input sequence length to the model is 49 and 196. So when setting $P = 1$ or $P = 2$, with such sequence lengths, when using an efficient attention engine, like Performer, which provides an approximation of the dot-product-attention, we do not gain a lot on the speed and we lose a bit of performance. However, when using a larger partition size, the additional compute with respect to the performance gain becomes reasonable.

In both /32 and /16, the computation cost is almost similar for $P = 4$ and $P = 6$. With $P = 4$, we have three omnidirectional attention, each applied on 4 layers, while with $P = 6$ we have two omnidirectional attention, each applied on 6 layers. However, $P = 6$ gives slightly better results in terms of accuracy and is placed on a sweeter spot in this trade-off. With $P = 12$, the computational costs of OmniNet increase, but the gain in the performance helps the model to be on the frontier of the compute-performance trade-off, when it is compared to OmniNet$_{B/32}$ and OmniNet$_{B/16}$.

4.6. Visualization

OmniNet combines information from different layers via two sequential mechanisms (§3.2.1): (1) omnidirectional attention, where representations of all tokens in all layers

get updated with respect to each other using an efficient attention mechanism; and (2) a pooling operation, where for each token, we collect the best values from all layers.

In order to understand how these two mechanisms combine information across different layers, we visualize attention maps (Abnar & Zuidema, 2020) and pooling statistics for a set of examples in the image recognition task. Figure 4 depicts three example inputs, where we show how OmniNet attends to different layers, as well as each layer's contribution during the pooling operation.

We can see that in some layers, attention seems to detect the objects in the image via attending to the edges or specific parts of the object, while in other layers, the attention mechanism uses mostly background information. It is clear that omnidirectional attention does indeed use such information by actively attending to layers of varying depth.

Additionally, when performing the element-wise pool operation over all the layers for each token, only a fraction of values from each layer's representation make it to the final representation. The bottom rows in Figure 4 illustrate this fraction for each token (image patch) across different layers. In most examples, we observe that a majority of the representation after the pooling operation comes from the first few layers. This is further evidence of how OmniNet can provide an explicit path for directing fine-grained information that is captured by the early layers to the final output, leading to much richer representations. For the sake of brevity, we refer readers to the Appendix for more detailed plots for these examples as well as other examples, which illustrate the same trends.

5. Conclusion

In this paper, we proposed OmniNet, which uses omnidirectional attention to connect all tokens across the entire network via self-attention. In order to manage the computational costs of the full receptive field, the meta-learner in OmniNet is parameterized by fast and efficient self-attention models. The proposed method achieves stellar performance on a myriad of language and vision tasks. Concretely, OmniNet achieves state-of-the-art performance on WMT EnDe and EnFr, outperforming deep 60-layer transformers. OmniNet also demonstrates substantial improvement over ViT on image recognition tasks.

References

Abnar, S. and Zuidema, W. Quantifying attention flow in transformers. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, 2020.

Baevski, A. and Auli, M. Adaptive input representations for neural language modeling. *arXiv preprint arXiv:1809.10853*, 2018.

Bahdanau, D., Cho, K., and Bengio, Y. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

Bapna, A., Chen, M. X., Firat, O., Cao, Y., and Wu, Y. Training deeper neural machine translation models with transparent attention. *arXiv preprint arXiv:1808.07561*, 2018.

Bradbury, J., Frostig, R., Hawkins, P., Johnson, M. J., Leary, C., Maclaurin, D., Necula, G., Paszke, A., VanderPlas, J., Wanderman-Milne, S., and Zhang, Q. JAX: composable transformations of Python+NumPy programs, 2018. URL http://github.com/google/jax.

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020.

Carion, N., Massa, F., Synnaeve, G., Usunier, N., Kirillov, A., and Zagoruyko, S. End-to-end object detection with transformers. *arXiv preprint arXiv:2005.12872*, 2020.

Chelba, C., Mikolov, T., Schuster, M., Ge, Q., Brants, T., Koehn, P., and Robinson, T. One billion word benchmark for measuring progress in statistical language modeling. *arXiv preprint arXiv:1312.3005*, 2013.

Chen, M., Radford, A., Child, R., Wu, J., Jun, H., Luan, D., and Sutskever, I. Generative pretraining from pixels. In *International Conference on Machine Learning*, pp. 1691–1703. PMLR, 2020.

Choromanski, K., Likhosherstov, V., Dohan, D., Song, X., Gane, A., Sarlos, T., Hawkins, P., Davis, J., Mohiuddin, A., Kaiser, L., et al. Rethinking attention with performers. *arXiv preprint arXiv:2009.14794*, 2020.

Dai, Z., Yang, Z., Yang, Y., Carbonell, J., Le, Q. V., and Salakhutdinov, R. Transformer-xl: Attentive language models beyond a fixed-length context. *arXiv preprint arXiv:1901.02860*, 2019.

Dehghani, M., Gouws, S., Vinyals, O., Uszkoreit, J., and Kaiser, Ł. Universal transformers. *arXiv preprint arXiv:1807.03819*, 2018.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

He, R., Ravula, A., Kanagal, B., and Ainslie, J. Realformer: Transformer likes residual attention. *arXiv e-prints*, pp. arXiv–2012, 2020.

He, T., Tan, X., Xia, Y., He, D., Qin, T., Chen, Z., and Liu, T.-Y. Layer-wise coordination between encoder and decoder for neural machine translation. In *Proceedings of the 32Nd International Conference on Neural Information Processing Systems*, pp. 7955–7965, 2018.

Heek, J., Levskaya, A., Oliver, A., Ritter, M., Rondepierre, B., Steiner, A., and van Zee, M. Flax: A neural network library and ecosystem for JAX, 2020. URL http://github.com/google/flax.

Huang, G., Liu, Z., Van Der Maaten, L., and Weinberger, K. Q. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4700–4708, 2017.

Kolesnikov, A., Beyer, L., Zhai, X., Puigcerver, J., Yung, J., Gelly, S., and Houlsby, N. Big transfer (bit): General visual representation learning. *arXiv preprint arXiv:1912.11370*, 2019.

Krizhevsky, A., Hinton, G., et al. Learning multiple layers of features from tiny images. 2009.

Kudo, T. and Richardson, J. Sentencepiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. *arXiv preprint arXiv:1808.06226*, 2018.

Kumar, M., Weissenborn, D., and Kalchbrenner, N. Colorization transformer. *arXiv preprint arXiv:2102.04432*, 2021.

Langley, P. Crafting papers on machine learning. In Langley, P. (ed.), *Proceedings of the 17th International Conference on Machine Learning (ICML 2000)*, pp. 1207–1216, Stanford, CA, 2000. Morgan Kaufmann.

Liu, X., Duh, K., Liu, L., and Gao, J. Very deep transformers for neural machine translation. *arXiv preprint arXiv:2008.07772*, 2020.

Nilsback, M.-E. and Zisserman, A. Automated flower classification over a large number of classes. In *2008 Sixth Indian Conference on Computer Vision, Graphics & Image Processing*, pp. 722–729. IEEE, 2008.

Ott, M., Edunov, S., Grangier, D., and Auli, M. Scaling neural machine translation. In *Proceedings of the Third Conference on Machine Translation: Research Papers*, pp. 1–9, Brussels, Belgium, October 2018. Association for Computational Linguistics. doi: 10.18653/v1/W18-6301. URL https://www.aclweb.org/anthology/W18-6301.

Parikh, A. P., Täckström, O., Das, D., and Uszkoreit, J. A decomposable attention model for natural language inference. *arXiv preprint arXiv:1606.01933*, 2016.

Parkhi, O. M., Vedaldi, A., Zisserman, A., and Jawahar, C. Cats and dogs. In *2012 IEEE conference on computer vision and pattern recognition*, pp. 3498–3505. IEEE, 2012.

Post, M. A call for clarity in reporting bleu scores. *arXiv preprint arXiv:1804.08771*, 2018.

Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., and Liu, P. J. Exploring the limits of transfer learning with a unified text-to-text transformer. *arXiv preprint arXiv:1910.10683*, 2019.

So, D. R., Liang, C., and Le, Q. V. The evolved transformer. *arXiv preprint arXiv:1901.11117*, 2019.

Srivastava, R. K., Greff, K., and Schmidhuber, J. Highway networks. *arXiv preprint arXiv:1505.00387*, 2015.

Sun, C., Shrivastava, A., Singh, S., and Gupta, A. Revisiting unreasonable effectiveness of data in deep learning era. In *Proceedings of the IEEE international conference on computer vision*, 2017.

Tay, Y., Tuan, L. A., Hui, S. C., and Su, J. Densely connected attention propagation for reading comprehension. *arXiv preprint arXiv:1811.04210*, 2018.

Tay, Y., Dehghani, M., Abnar, S., Shen, Y., Bahri, D., Pham, P., Rao, J., Yang, L., Ruder, S., and Metzler, D. Long range arena: A benchmark for efficient transformers. *arXiv preprint arXiv:2011.04006*, 2020a.

Tay, Y., Dehghani, M., Bahri, D., and Metzler, D. Efficient transformers: A survey. *arXiv preprint arXiv:2009.06732*, 2020b.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. In *Advances in neural information processing systems*, pp. 5998–6008, 2017.

Wang, S., Li, B., Khabsa, M., Fang, H., and Ma, H. Linformer: Self-attention with linear complexity. *arXiv preprint arXiv:2006.04768*, 2020.

Zaheer, M., Guruganesh, G., Dubey, A., Ainslie, J., Alberti, C., Ontanon, S., Pham, P., Ravula, A., Wang, Q., Yang, L., et al. Big bird: Transformers for longer sequences. *arXiv preprint arXiv:2007.14062*, 2020.

A. Detailed Experimental Setup

This section describes several details of our experiments.

A.1. Dataset Specific Setups

For all experiments, we implement code using Python and JAX. Specifically, the main Transformer blocks and codebase for most experiments are derived from FLAX examples. For WMT'17, we build sentencepiece tokenizers of $32K$ from the dataset. WMT'17 collections are obtained from Tensorflow datasets (TFDS). For autoregressive language modeling, the C4 corpus is similarly found in TFDS. For both LM1B and C4 tasks, we use a sentencepiece vocab of $32K$ subwords.

A.2. Efficient Transformer Hyperparameters

For Xformers (efficient transformers), we use implementations derived from FLAX[5].For Linformer, we use $k = 32$ for the low-rank down projection with shared parameters for both key and value. For Performer, we use the default setup from the official implementation. This corresponds to the generalized attention with ReLU activations. We do not use any random features. For BigBird, our codebase similarly links to the official implementation and use the default hyperparameters. The block size is 64 for BigBird and the number of random blocks is 3.

B. Visualisation of Contributions of Layers in Omnidirectional Representations

Figures 5 to 9 (in subsequent pages) show contributions of different layers in omnidirectional representations in terms of detailed attention maps (attention distribution over all layers, in all heads, when the CLS token in the omnidirectional layer is considered as the query) as well as contribution of different layers in the pooling operation.

---

[5]https://github.com/google/flax.
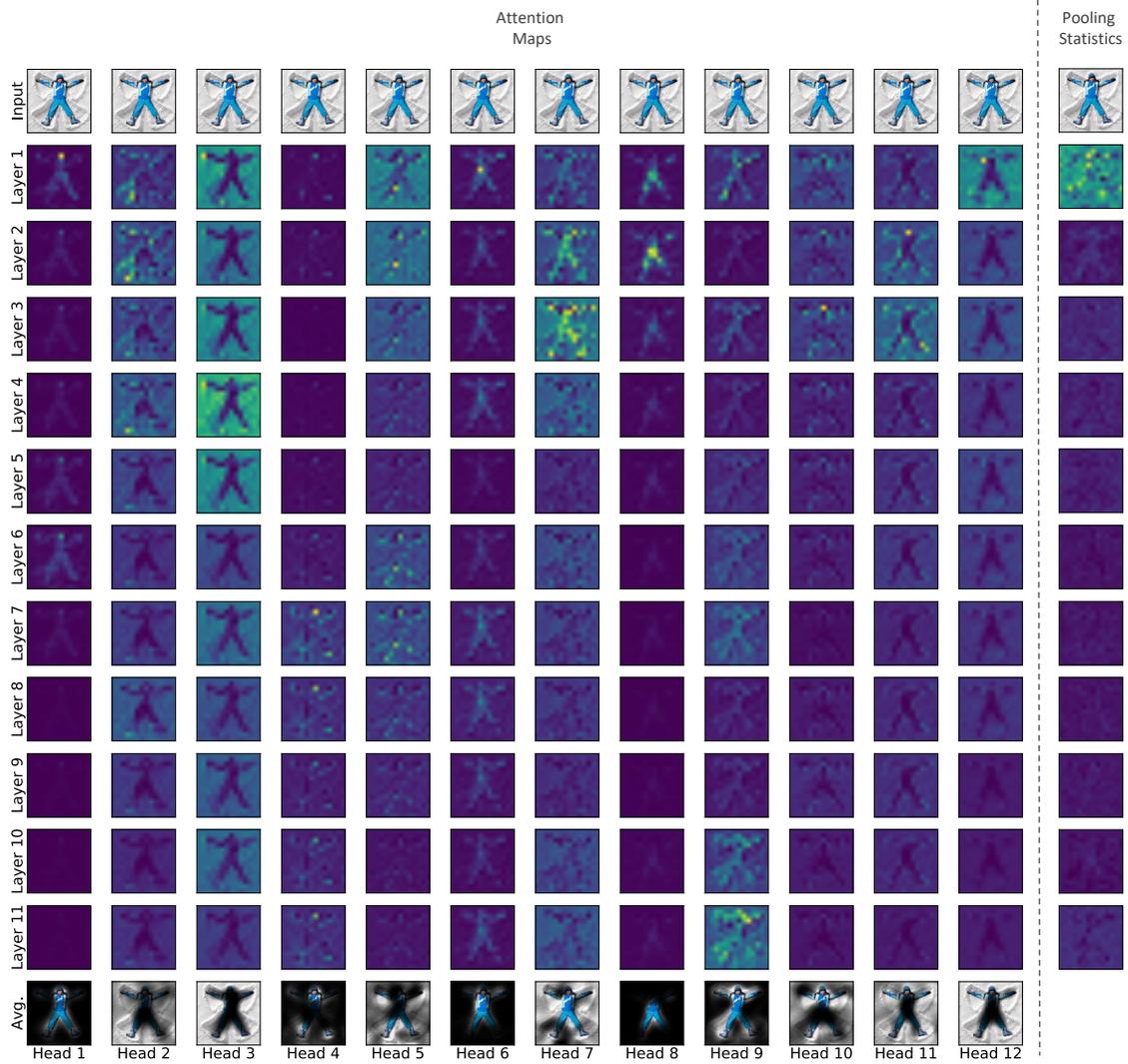
Figure 5. Contributions of different layers in omnidirectional representations for Example #1.
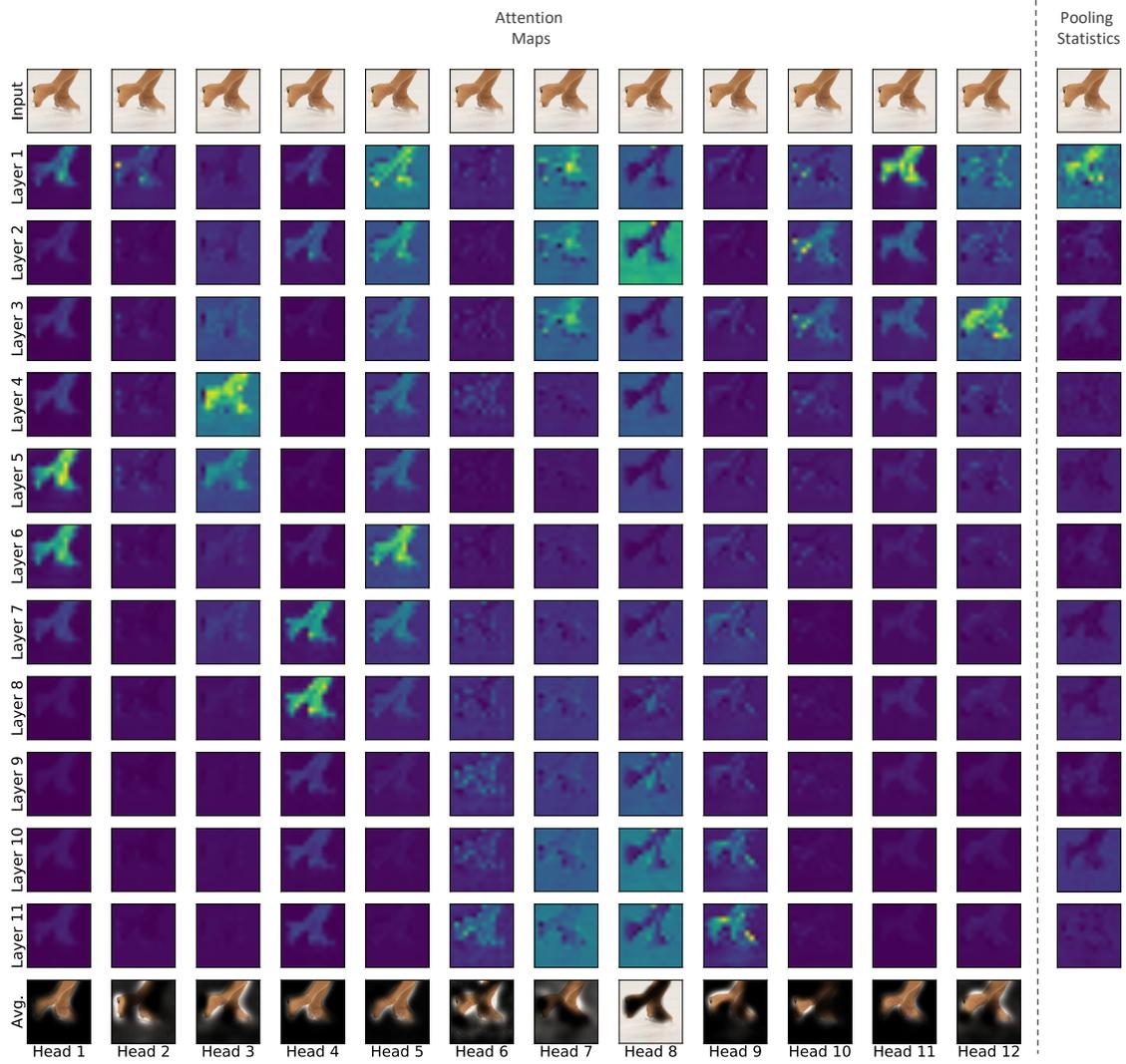
Figure 6. Contributions of different layers in omnidirectional representations for Example #2.
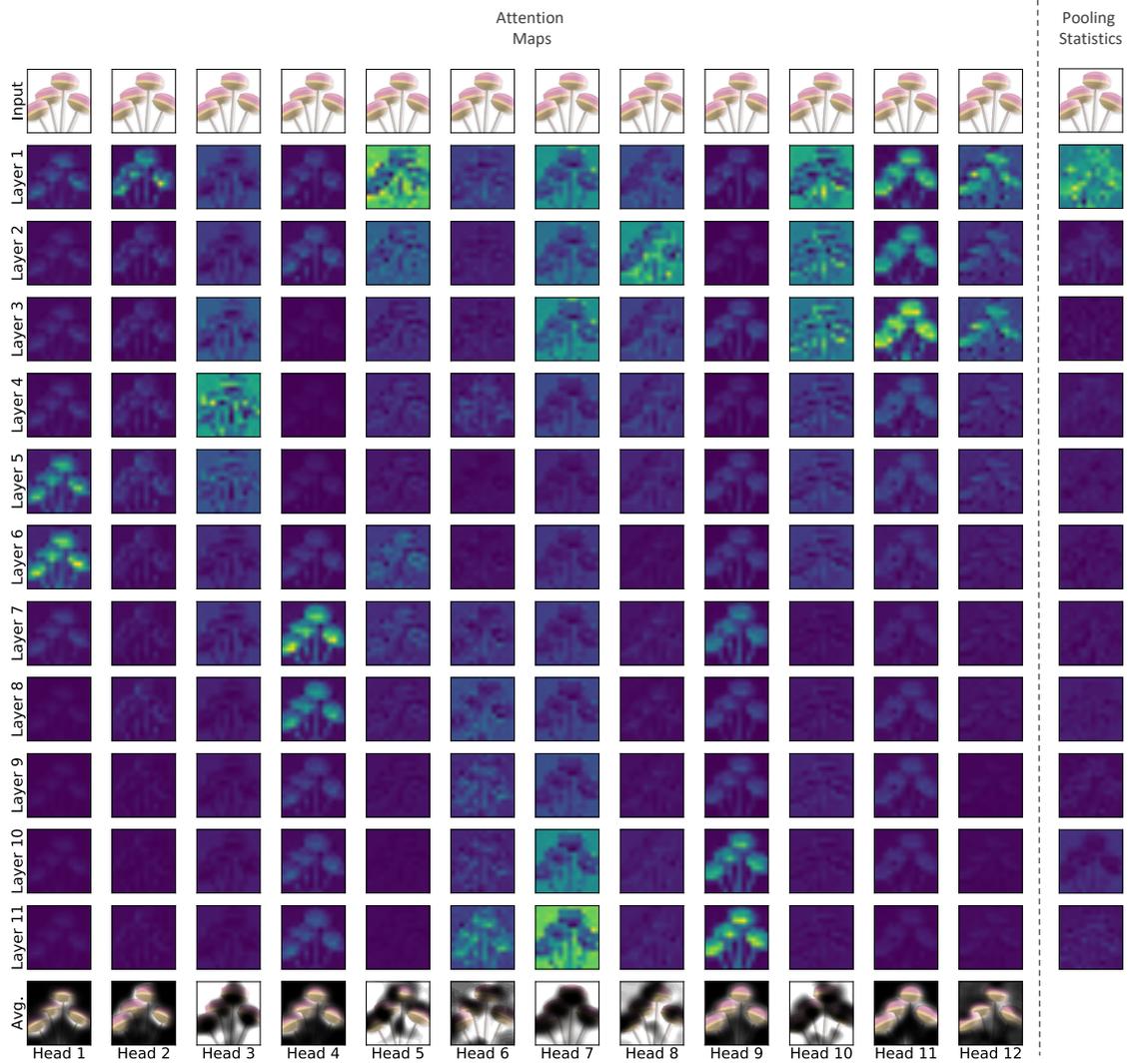
Figure 7. Contributions of different layers in omnidirectional representations for Example #3.

*Figure 8.* Contributions of different layers in omnidirectional representations for Example #4.
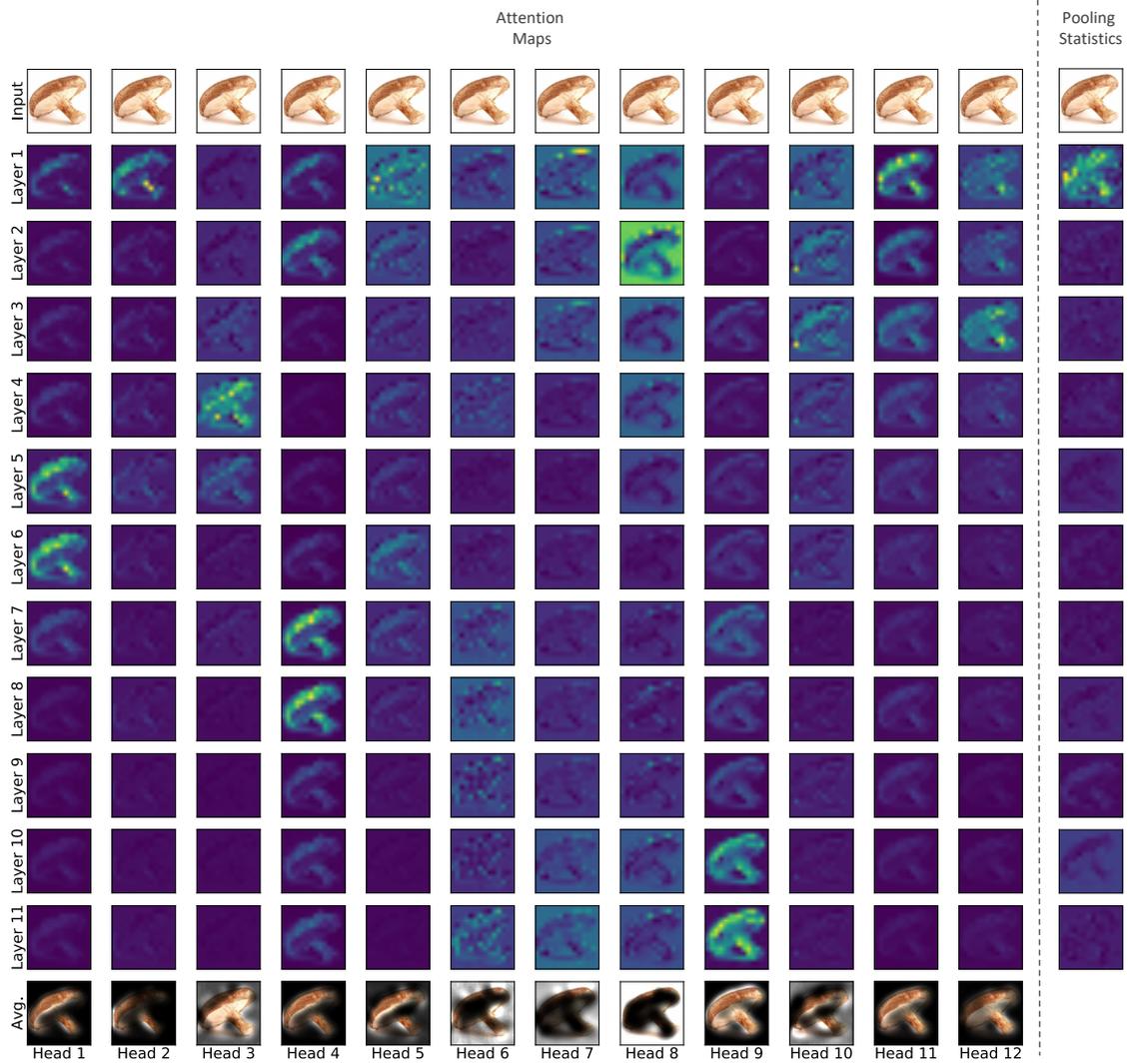
Figure 9. Contributions of different layers in omnidirectional representations for Example #5.