Supplemental material: Learned Queries for Efficient Local Attention

Moab Arar
Tel-Aviv University

Ariel Shamir
Reichman University

Amit H. Bermano
Tel-Aviv University

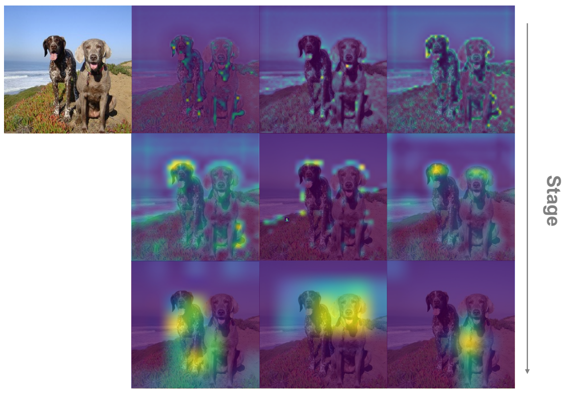

Figure 1. **QnA attention visualization of different heads.** To visualize a specific location's attention score, we sum the attention scores obtained for that location in all windows. Attention maps are upsampled and overlaid on top of the image for better visualization.

1. Attention visualization

In QnA, the aggregation kernel of each window is derived from the attention scores between the learned queries and the window keys. To visualize the attention of the whole image, we choose to sum the scores of each location as obtained in all relevant windows. You can find the visualization in Figure 1. As shown in Figure 1, the attentions are content-aware, suggesting the window aggregation kernels are spatially adaptive. For the learned kernels in each window, please refer to Figure 2.

2. Full training details

2.1. Image Classification

We evaluate our method using the ImageNet-1K [24] benchmark, which contains 1.28M training images and 50,000 validation images from 1,000 classes. We follow the training recipe of DEiT [26], except we omit EMA [22] and repeated augmentations [11]. Particularly, we train all models for 300-epochs, using the AdmaW [15, 20] optimizer. We employ a linearly scaled learning rate $lr = \texttt{5e-4} \cdot \frac{\text{Batch size}}{256}$ [9], with warmup epochs [19] varying according to model size and weight decay $wd = \texttt{5e-2}$. For augmentations, we apply RandAugment [6], mixup [33] and CutMix [32] with label-smoothing [31], and color-jitter [5]. Finally, an increasing stochastic depth is applied [14]. Note this training recipe (with minor discrepancy between previous papers), is becoming the standard when training a Vision Transformer on the ImageNet benchmark. Finally, we normalize the queries in all QnA layers to be unit-vectors for better training stability.

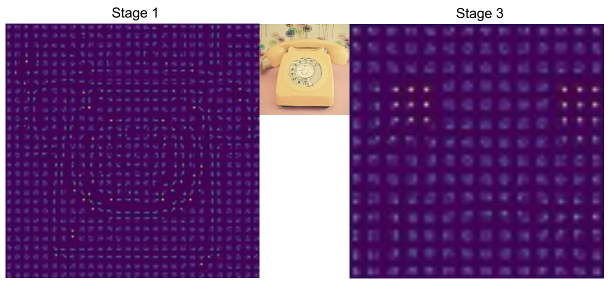

Figure 2. **QnA aggregation kernels visualization.** The attention kernels are tiled in the visualization, instead of overlapped, causing the uniform grid effect. Brighter areas indicate higher attention scores. (best viewed when zoomed-in).

2.2. Object Detection

We train the DETR model on COCO 2017 detection dataset [17], containing $118K$ training images and a $5k$ validation set size. We utilize the training setting of DETR, in which the input is resized such that the shorter side is between 480 and 800 while the longer side is at most 1333 [29]. An initial learning rate of $\texttt{1e}-4$ is set for the detection transformer, and $\texttt{1e}-5$ learning rate for the backbone network. Due to computational limitations, we use a short training scheduler of 75 epochs with a batch size of 32. The learning rates are scaled by 0.1 after 50 epochs. We trained DETR using the implementation in [7].

3. QnA-ViT architecture

The QnA-ViT architecture is composed of transformer blocks [8] and QnA blocks. First we split the image into

4×4 patches, and project them to form the input tokens. The vision transformer block is composed of a multihead attention layer (MSA) and an inverted-bottleneck feed forward network (FFN), with expansion 4. The output of block-l is:

$$z_l' = \mathbf{MSA}\left(\mathbf{LayerNorm}(z_{l-1})\right) + z_{l-1}$$
$$z_l = \mathbf{FFN}\left(\mathbf{LayerNorm}(z_l')\right) + z_l'.$$

The QnA block shares a similar structure, except we replace the MSA layer with QnA layer. Downampling is performed using QnA blocks with stride set to 2 (to enable skip-connections we use 1×1-convolution with similar stride). We employ pre-normalization [30] to stabilize training. Finally, we use global average pooling [16] right before the classification head, with LayerNorm [1] employed prior to the pooling operation. Full architecture details can be found in Table 1.

4. Implementation & complexity - extended

In this section we provide full details on the efficient implementation of QnA. To simplify the discussion, we only consider a single-query without positional embedding.

Let us first examine the output of a QnA layer, by expanding the softmax operation inside the attention layer:

$$z_{i,j} = \mathbf{Attention}\left(\tilde{q}, K_{\mathcal{N}_{i,j}}\right) \cdot V_{\mathcal{N}_{i,j}} = \frac{\sum_{\mathcal{N}_{i,j}} e^{\tilde{q}K_{n,m}} v_{n,m}}{\sum_{\mathcal{N}_{i,j}} e^{\tilde{q}K_{n,m}}}.$$

Recall, $\mathcal{N}_{i,j}$ is the $k \times k$-window at location (i, j). While it may seem that we need to calculate the query-key dot product for each window, notice that since we use the same query over each window, then we can calculate $\tilde{q}K^T$ once for the entire input. Also, we can leverage the matrix multiplication associativity and improve the computation complexity by calculating $\tilde{q}W_k^T$ first (this fused implementation reduces the memory by avoiding the allocation of the key entities). Once we calculate the query-key dot product, we can efficiently aggregate the dot products using the sum-reduce operation supported in many deep learning frameworks (e.g., Jax [3]). More specifically, let $\mathbf{Sum}_k(\dots)$ be a function that sums-up values in each $k \times k$-window, then:

$$z_{i,j} = \frac{\sum_{\mathcal{N}_{i,j}} e^{\tilde{q}K_{n,m}} v_{n,m}}{\sum_{\mathcal{N}_{i,j}} e^{\tilde{q}K_{n,m}}} = \frac{\mathbf{Sum}_k\left(e^{\tilde{q}K^T} * V\right)}{\mathbf{Sum}_k\left(e^{\tilde{q}K^T}\right)},$$

where * is the element wise multiplication. A pseudo-code of our method can be found in Algorithm 1. We further provide a code-snippet of the QnA module, implemented in Jax/Flax [4, 10] (see Figure 3).

Complexity Analysis: in the single query variant, extracting the values and computing the key-query dot product require $2HWD^2$ computation and $HW + HWD$ extra space. Additionally, computing the softmax using the above method requires additional $\mathcal{O}(k^2HWD)$ computation (for summation and division), and $\mathcal{O}(HWD)$ space (which is independent of k, i.e., the window size). For multiple-queries variant ($L = 2$), see empirical comparison in the main paper.

Algorithm 1 Efficient implementation of QnA. All operations can be implemented efficiently with little memory-overhead. Further, $\mathbf{Sum}_k$ applies sum-reduction to all elements in each $k \times k$-window.

```
Input: $X \in \mathbb{R}^{H \times W \times D}$
Parameters: $W_K, W_V \in \mathbb{R}^{D \times D}$, $\tilde{q} \in \mathbb{R}^D$
   // Compute values:
1:    $V \leftarrow XW_V$
   // Compute the query-key dot product ( $\mathbf{S} \leftarrow \mathbf{\tilde{q}K^T}$):
2:    Let $A \leftarrow \tilde{q}W_K^T$
3:    for $l \in [L], i, j \in [H] \times [W]$ do
4:        $S_{l,i,j} \leftarrow A_{l,i,j} \cdot X_{i,j}^T$
5:    end for
   // Compute the final output:
6:    Let $B \leftarrow e^S$ be the element-wise exponent of $S$
7:    Let $C \leftarrow B * V$      ▷ $*$ is the element-wise product
8:    return $\mathbf{Sum}_k(C)/\mathbf{Sum}_k(B)$
```

5. Design choices - full report

5.1. Number of queries

To verify the effectiveness of using multiple queries, we trained a lightweight QnA-ViT network composed of *local self-attention blocks* and *QnA blocks*. We set the window size of all local self-attention layers to be 7x7, and we use a 3x3 receptive field for QnA layers. All the downsampling performed are QnA-Based. A similar architecture was used for the SASA [21] baseline, where we replaced the QnA layers with SASA layers. The number of QnA/SASA blocks used for each stage are $[2, 2, 2, 0]$ and the number of local self-attention blocks are $[1, 1, 5, 2]$.

5.2. Number of heads

As stated in the main text, using more attention heads is beneficial for QnA. More specifically, we conducted two experiments, one where we use only QnA blocks and the second where we use both QnA blocks and self-attention blocks. In the first experiment, we use the standard ImageNet training preprocessing [25], meaning we employ random crop with resize and random horizontal flip. In the second experiment, we used DeiT training preprocessing. We show the full report in Table 2 and Table 3.

First, from Table 2, we notice that training shallow QnA-networks for fewer epochs requires many attention heads.

Stage	Output	QnA-T		QnA-S		QnA-B	
		QnA Blocks	SA Blocks	QnA Blocks	SA Blocks	QnA Blocks	SA Blocks
1	56x56	4x4 Conv, stride 4, dim 64		4x4 Conv, stride 4, dim 64		4x4 Conv, stride 4, dim 96	
		[3x3 QnA-Block, stride 1, head 8] × 2	None	[3x3 QnA-Block, stride 1, head 8] × 2	None	[3x3 QnA-Block, stride 1, head 6] × 2	None
2	28x28	3x3 QnA, stride 2, head 16, dim 128		3x3 QnA, stride 2, head 16, dim 128		3x3 QnA, stride 2, head 16, dim 192	
		[3x3 QnA-Block, stride 1, head 16] × 3	None	[3x3 QnA-Block, stride 1, head 16] × 3	None	[3x3 QnA-Block, stride 1, head 12] × 3	None
3	14x14	3x3 QnA, stride 2, head 32, dim 256		3x3 QnA, stride 2, head 32, dim 256		3x3 QnA, stride 2, head 32, dim 384	
		[3x3 QnA-Block, stride 1, head 32] × 2	[SA-Block, win sz. 14 x 14, head 8] × 4	[3x3 QnA-Block, stride 1, head 32] × 6	[SA-Block, win sz. 14 x 14, head 8] × 12	[3x3 QnA-Block, stride 1, head 24] × 6	[SA-Block, win sz. 14 x 14, head 12] × 12
4	7x7	3x3 QnA, stride 2, head 64, dim 512		3x3 QnA, stride 2, head 64, dim 512		3x3 QnA, stride 2, head 48, dim 768	
		None	[SA-Block, win sz. 7 x 7, head 16] × 2	None	[SA-Block, win sz. 7 x 7, head 16] × 2	None	[SA-Block, win sz. 7 x 7, head 24] × 2

Table 1. **QnA-ViT architecture details.** QnA is used to down-sample the feature maps between two consecutive stages. In stage 3 we first employ global self-attention blocks.

```
class QnA(nn.Module):
  @nn.compact
  def __call__(self, X):
    # Initialize Parameters:
    Q  = # Query vectors  [L, h, D//h]
    Wk = # Linear Projection for keys [D, D]
    Ws = # Attention weight scale [k, k, L * h]
    B_rpe = #Relative PE [k, k, L * h]
    # Fused implementation of Q*(X*W_K).Transpose
    Wk = Wk.reshape([-1, heads, D // heads])
    QWk = jnp.einsum('lhd,Dhd->Dlh', Q, Wk)
    QK_similariy = jnp.einsum('BHWD,BDqh->BHWqh', X, QWk)
    # Compute V
    V = nn.Dense(D)(X).reshape([B, H, W, heads, D // heads])
    # Compute Attention:
    exp_similarity = jnp.exp(QK_similariy)  # [B, H, W, L, h]
    exp_similarity_v = exp_similarity[..., jnp.newaxis] * V[:,:,: jnp.newaxis,:]  # [B, H, W, L, D]
    aux_kernel = (jnp.exp(B_rpe) * Ws).repeat(repeats=D // heads, axis=-1)  # [k, k, d, L*h]
    A = jax.lax.conv_general_dilated(exp_similarity_v.reshape([B, H, W, LD]),
                            aux_kernel, window_strides=[s, s], padding='SAME',
                            feature_group_count=L * D,).reshape([B, H_out, W_out, L, heads, D // heads
    ])
    A = jnp.reshape(A, [B, H_out, W_out, L, heads, D // heads])
    aux_kernel = jnp.exp(B_rpe)  # [k, k, 1, L*h]
    B = jax.lax.conv_general_dilated(exp_similarity.reshape([B, H, W, -1]),
                                     aux_kernel,
                                     window_strides=[s, s],
                                     padding='SAME',
                                     dimension_numbers=_conv_dimension_numbers(I.shape)
                                     ).reshape([B, H_out, W_out, L, heads, 1])
    out_heads = jnp.sum(A / B, axis=-2).reshape([B, H_out, W_out, D])
    final_out = nn.Dense(self.D)(out_heads)
    return final_out
```

Figure 3. **Code snippet of the QnA module implemented in Jax [4] and Flax [10].**

Furthermore, it is better to maintain a fixed dimension head across stages - this is done by doubling the number of heads between two consecutive stages. For deeper networks, the advantage of using more heads becomes less significant. This is because the network can capture more feature subspaces by leveraging its additional layers. Finally, when using both QnA and ViT blocks, it is still best to use more heads for QnA layers, while for ViT blocks, it is best to use a high dimension representation by having fewer heads (see Table 3).

5.3. How much QnA do you really need?

To understand the benefit of using QnA layers, we consider a dozen network architectures that combine vanilla

QnA Blocks	Heads	AugReg	Epochs	Top-1 Acc.
[2,2,2,2]	[2,2,2,2]	None	90	68.33
	[4,4,4,4]			69.05
	[8,8,8,8]			69.98
	[2,4,8,16]			70.08
	[4,8,16,32]			70.54
	[8,16,32,64]			**71.12**
[3,4,6,3]	[2,4,8,16]	None	90	72.66
	[4,8,16,32]			72.82
	[8,16,32,64]			**73.02**

Table 2. **The affect of number of heads on QnA.** We train two networks using the Inception preprocessing [25], i.e., random crop and horizontal flip. We set the number of QnA blocks according to the ResNet-18 and ResNet-50 networks (the number of blocks for each stage is stated in the first column). As can be seen, using fixed head-dimension is better than increasing the head-dimension as we propagate through the network. Shallow networks benefit from having many heads, while deeper networks gain less from more heads. Therefore we suggest increasing the head-dimension for deeper networks for better memory utilization.

Blocks		Heads		AugReg	Epochs	Top-1 Acc.
QnA	SA	QnA	SA			
[1,2,3,0]	[1,1,3,2]	[2,4,8,16]	[2,4,8,16]	DeiT [26]	90	78.17
		[4,8,16,32]	[4,8,16,32]			79.24
		[8,16,32,64]	[8,16,32,64]			79.34
		[8,16,32,64]	[2,4,8,16]			79.31
		[8,16,32,64]	[4,8,16,32]			**79.53**
[1,2,3,0]	[1,1,3,2]	[8,16,32,64]	[2,4,8,16]	DeiT [26]	300	**81.5**
		[8,16,32,64]	[4,8,16,32]			81.49

Table 3. **The number of heads affect on QnA and ViT Blocks.** QnA still benefits from more heads, while ViT blocks need higher-dimension representation, specially for longer training.

ViT and QnA blocks. For ViT blocks, we tried to use global attention in the early stages but found it better to use local self-attention and restrict the window size to be at most 14x14. We group the architecture choices into three groups:

1. We consider varying the number of QnA blocks in the early stages.

2. We change the number of QnA blocks in the third stage.

3. We use lower window size for the local-self attention blocks. Namely, 7x7 window in all ViT blocks at all stages.

Finally, all networks were trained for 300 epochs following DeiT preprocessing. The full report can be found in Table 4.

From Table 4, local-self attention is not very beneficial in the early stages and can be omitted by using QnA blocks only. Furthermore, it is better to use more ViT blocks in deeper stages, but we can still reduce the network's latency by having some QnA blocks. Finally, when using a lower window size, the local self-attention becomes less effective. Moreover, since QnA is shift-invariant, it can mitigate the lack of cross-window interactions, reflected in the improvement gain we achieve when using more QnA blocks.

Blocks		Params	GFLOPS	Top- Acc
QnA	SA			
Changes in stages 1, 2				
[1,1,4,0]	[3,3,3,2]	16.62M	3.200	81.70
[2,2,4,0]	[2,2,3,2]	16.51M	2.909	81.74
[3,3,4,0]	[1,1,3,2]	16.40M	2.875	**81.86**
[4,4,4,0]	[0,0,3,2]	**16.30M**	**2.584**	81.83
Changes in stage 3				
[4,4,7,0]	[0,0,0,2]	16.00M	2.631	80.8
[4,4,5,0]	[0,0,2,2]	16.25M	2.698	81.30
[4,4,3,0]	[0,0,4,2]	**16.40M**	**2.628**	**81.9**
[4,4,1,0]	[0,0,6,2]	16.55M	2.714	**81.9**
Local SA with window size 7×7				
[1,1,1,0]	[2,2,6,2]	16.38M	2.568	80.0
[2,2,2,0]	[1,1,5,2]	16.3M	2.491	80.6
[2,2,3,0]	[1,1,4 ,2]	16.23M	2.471	80.7
[2,2,4,0]	[1,1,3, 2]	**16.16M**	**2.450**	**80.8**

Table 4. **How much QnA do you really need? - full report.** The number of QnA and local self-attention (SA) blocks in each stage are indicated in the first row. In the first two sub-tables, a window size of 14×14 except in the last stage, where a 7×7 window size was set. In the last sub-table, we reduce the window size to become 7×7 for all stages.

6. QnA variants - extended

As discussed in the paper, we incorporate multi-head attention and positional embedding in our layer.

Positional Embedding self-attention is a permutation invariant operation, meaning, it doesn't assume any spatial relations between the input tokens. This property is not desirable in image processing, where relative-context is important. To alleviate this, position encoding can be injected into the self-attention mechanism. Following recent literature, we use relative-positional embedding [2,12,13,23,28]. This introduces a spatial bias into the attention scheme, rendering Equation 3 (from the main text) now to be:

$$\textbf{Attention}\left(Q,K\right) = \textbf{Softmax}\left(QK^T/\sqrt{D} + B\right), \quad (1)$$

where $B \in \mathbb{R}^{k\times k}$ is a learned relative positional encoding. Note, different biases are learned for each query in the QnA layer, which adds $\mathcal{O}(L \times k^2)$ additional space.

Multi-head attention: as in the original self-attention layer [27], we use multiple heads in order to allow the QnA layer to attend different features simultaneously. In fact, as we will show in Section 5.2, using more attention heads is beneficial to QnA.

In mutli-head attention, all queries Q, keys K, and values V entities are split into h sub-tensors, which will correspond to vectors in the lower dimensional space $\mathbb{R}^{D/h}$. More specifically, let $Q^{(i)}, K^{(i)}, V^{(i)}$ be the i-th sub vector of each query, key and value, respectively, the self-attention of head-i becomes:

$$\begin{aligned} \text{head}_i &= \textbf{Attention}\left(Q^{(i)}, K^{(i)}\right) V^{(i)} \\ &= \textbf{Softmax}\left(Q^{(i)} K^{(i)T} / \sqrt{d_h} + B\right) V^{(i)}, \end{aligned} \tag{2}$$

, where $d_h = D/h$, and the output of the Multi-Head Attention (MHA) is:

$$\begin{aligned} &\textbf{MHA}\left(Q, K, V\right) = \\ &\qquad \texttt{Concat}\left(\text{head}_1, \ldots, \text{head}_h\right) W_O \end{aligned} \tag{3}$$

, where $W_O \in \mathbb{R}^{D \times D}$ is the final projection matrix.

Upsampling via QnA as stated in the paper, upsampling by factor s can be defined using s^2-learned queries, i.e., $\tilde{Q} \in \mathbb{R}^{s^2 \times D}$. To be precise, the output of each window $\mathcal{N}_{i,j}$ is expressed via:

$$z_{i,j} = \textbf{Attention}\left(\tilde{Q}, K_{\mathcal{N}_{i,j}}\right) \cdot V_{\mathcal{N}_{i,j}}. \tag{4}$$

Note, the window output given by Eq. (4) is now a matrix of size $s^2 \times D$, and the total output Z is a tensor of size $H \times W \times s^2 \times D$. To form the upsampled output Z^s, we need to reshape and permute the tensor's axes:

$$\begin{aligned} Z^s_{(1)} &= \text{Reshape}(Z, [H, W, s, s, D]) \\ Z^s_{(2)} &= \text{Permute}(Z^s_{(1)}, [0, 2, 1, 3, 4]) \\ Z^s &= \text{Reshape}(Z^s_{(2)}, [H \times s, W \times s, D]). \end{aligned} \tag{5}$$

7. QnA as an upsampling layer

In the paper we showed how QnA can be used as an upsampling layer. In particular, we trained a simple auto-encoder network, composed of five downsampling layers and 5 upsampling layers. We use the L_1 reconstruction loss as an objective function to train the auto-encoder. We considered three different encoder layers:

- **ConvS2-IN**: 3x3 convolution with stride 2 followed by Instance Normalization layers
- **Conv-IN-Max**: 3x3 convolution with stride 1 followed by Instance Normalization layer and max-pooling with stride 2
- **LN-QnA**: Layer norm followed by a 3x3 single-query QnA layer

and three different decoder layers:

- **Bilinear-Conv-IN**: x2 bilinear up-sampling followed by 3x3 convolution and instance normalization
- **ConvTranspose-IN**: a 2d transposed convolution followed by Instance Norm
- **LN-UQnA**: Layer Norm followed by a 3x3 upsampling QnA layer

For our baseline networks, we found it best to use **Conv-In-Max** in the encoder path, and chose either **Bilinear-Conv-In** or **ConvTranspose-IN** for the decoder path. For QnA based auto-encoders, we use QnA layers for both downsampling and upsampling.

We use the CelebA dataset [18], where all images are center-aligned and resized to resolution 256^2. All networks were trained for 10-epochs, using the Adam [15] optimizer (learning rates were chosen according to the best test-loss).

References

[1] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization, 2016. 2

[2] Hangbo Bao, Li Dong, Furu Wei, Wenhui Wang, Nan Yang, Xiaodong Liu, Yu Wang, Jianfeng Gao, Songhao Piao, Ming Zhou, and Hsiao-Wuen Hon. Unilmv2: Pseudo-masked language models for unified language model pre-training. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, volume 119 of *Proceedings of Machine Learning Research*, pages 642–652. PMLR, 2020. 4

[3] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: composable transformations of Python+NumPy programs, 2018. 2

[4] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: composable transformations of Python+NumPy programs, 2018. 2, 3

[5] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey E. Hinton. A simple framework for contrastive learning of visual representations. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, volume 119 of *Proceedings of Machine Learning Research*, pages 1597–1607. PMLR, 2020. 1

[6] Ekin Dogus Cubuk, Barret Zoph, Jon Shlens, and Quoc Le. Randaugment: Practical automated data augmentation with a reduced search space. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020. 1

[7] Mostafa Dehghani, Alexey Gritsenko, Anurag Arnab, Matthias Minderer, and Yi Tay. Scenic: A JAX library for computer vision research and beyond. *arXiv preprint arXiv:2110.11403*, 2021. 1

[8] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021. 1

[9] Priya Goyal, Piotr Dollár, Ross B. Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, large minibatch SGD: training imagenet in 1 hour. *CoRR*, abs/1706.02677, 2017. 1

[10] Jonathan Heek, Anselm Levskaya, Avital Oliver, Marvin Ritter, Bertrand Rondepierre, Andreas Steiner, and Marc van Zee. Flax: A neural network library and ecosystem for JAX, 2020. 2, 3

[11] Elad Hoffer, Tal Ben-Nun, Itay Hubara, Niv Giladi, Torsten Hoefler, and Daniel Soudry. Augment your batch: Improving generalization through instance repetition. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8129–8138, 2020. 1

[12] Han Hu, Jiayuan Gu, Zheng Zhang, Jifeng Dai, and Yichen Wei. Relation networks for object detection. In *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*, pages 3588–3597. Computer Vision Foundation / IEEE Computer Society, 2018. 4

[13] Han Hu, Zheng Zhang, Zhenda Xie, and Stephen Lin. Local relation networks for image recognition. In *2019 IEEE/CVF International Conference on Computer Vision, ICCV 2019, Seoul, Korea (South), October 27 - November 2, 2019*, pages 3463–3472. IEEE, 2019. 4

[14] Gao Huang, Yu Sun, Zhuang Liu, Daniel Sedra, and Kilian Q. Weinberger. Deep networks with stochastic depth. In Bastian Leibe, Jiri Matas, Nicu Sebe, and Max Welling, editors, *Computer Vision - ECCV 2016 - 14th European Conference, Amsterdam, The Netherlands, October 11-14, 2016, Proceedings, Part IV*, volume 9908 of *Lecture Notes in Computer Science*, pages 646–661. Springer, 2016. 1

[15] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. 1, 5

[16] Min Lin, Qiang Chen, and Shuicheng Yan. Network in network. In Yoshua Bengio and Yann LeCun, editors, *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*, 2014. 2

[17] Tsung-Yi Lin, Michael Maire, Serge J. Belongie, Lubomir D. Bourdev, Ross B. Girshick, James Hays, Pietro Perona, Deva Ramanan, Piotr Doll'a r, and C. Lawrence Zitnick. Microsoft COCO: common objects in context. *CoRR*, abs/1405.0312, 2014. 1

[18] Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. Deep learning face attributes in the wild. In *ICCV*, pages 3730–3738. IEEE Computer Society, 2015. 5

[19] Ilya Loshchilov and Frank Hutter. SGDR: stochastic gradient descent with warm restarts. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017. 1

[20] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. 1

[21] Niki Parmar, Prajit Ramachandran, Ashish Vaswani, Irwan Bello, Anselm Levskaya, and Jon Shlens. Stand-alone self-attention in vision models. In Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 68–80, 2019. 2

[22] Boris T Polyak and Anatoli B Juditsky. Acceleration of stochastic approximation by averaging. *SIAM journal on control and optimization*, 30(4):838–855, 1992. 1

[23] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *J. Mach. Learn. Res.*, 21:140:1–140:67, 2020. 4

[24] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015. 1

[25] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott E. Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA, June 7-12, 2015*, pages 1–9. IEEE Computer Society, 2015. 2, 4

[26] Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Hervé Jégou. Training data-efficient image transformers & distillation through attention. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, volume

139 of *Proceedings of Machine Learning Research*, pages 10347–10357. PMLR, 2021. 1, 4

[27] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 5998–6008, 2017. 5

[28] Kan Wu, Houwen Peng, Minghao Chen, Jianlong Fu, and Hongyang Chao. Rethinking and improving relative position encoding for vision transformer. *CoRR*, abs/2107.14222, 2021. 4

[29] Yuxin Wu, Alexander Kirillov, Francisco Massa, Wan-Yen Lo, and Ross Girshick. Detectron2. https://github.com/facebookresearch/detectron2, 2019. 1

[30] Ruibin Xiong, Yunchang Yang, Di He, Kai Zheng, Shuxin Zheng, Chen Xing, Huishuai Zhang, Yanyan Lan, Liwei Wang, and Tie-Yan Liu. On layer normalization in the transformer architecture. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, volume 119 of *Proceedings of Machine Learning Research*, pages 10524–10533. PMLR, 2020. 2

[31] Li Yuan, Francis E. H. Tay, Guilin Li, Tao Wang, and Jiashi Feng. Revisiting knowledge distillation via label smoothing regularization. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2020, Seattle, WA, USA, June 13-19, 2020*, pages 3902–3910. Computer Vision Foundation / IEEE, 2020. 1

[32] Sangdoo Yun, Dongyoon Han, Sanghyuk Chun, Seong Joon Oh, Youngjoon Yoo, and Junsuk Choe. Cutmix: Regularization strategy to train strong classifiers with localizable features. In *2019 IEEE/CVF International Conference on Computer Vision, ICCV 2019, Seoul, Korea (South), October 27 - November 2, 2019*, pages 6022–6031. IEEE, 2019. 1

[33] Hongyi Zhang, Moustapha Cissé, Yann N. Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018. 1