Visual Transformers: Where Do Transformers Really Belong in Vision Models?

Bichen Wu¹, Chenfeng Xu², Xiaoliang Dai¹, Alvin Wan², Peizhao Zhang¹, Zhicheng Yan¹, Masayoshi Tomizuka², Joseph Gonzalez², Kurt Keutzer², Peter Vajda¹

¹ Facebook Inc, ² University of California, Berkeley

{wbc, xiaoliangdai, stzpz, zyan3, vajdap}@fb.com,
{xuchenfeng, alvinwan, tomizuka, jegonzal, keutzer}@berkeley.edu

Abstract

A recent trend in computer vision is to replace convolutions with transformers. However, the performance gain of transformers is attained at a steep cost, requiring GPU years and hundreds of millions of samples for training. This excessive resource usage compensates for a misuse of transformers: Transformers densely model relationships between its inputs – ideal for late stages of a neural network, when concepts are sparse and spatially-distant, but extremely inefficient for early stages of a network, when patterns are redundant and localized. To address these issues, we leverage the respective strengths of both operations, building convolution-transformer hybrids. Critically, in sharp contrast to pixel-space transformers, our Visual Transformer (VT) operates in a semantic token space, judiciously attending to different image parts based on context. Our VTs significantly outperform baselines: On ImageNet, our VT-ResNets outperform convolution-only ResNet by 4.6 to 7 points and transformer-only ViT-B by 2.6 points with 2.5× fewer FLOPs, 2.1× fewer parameters. For semantic segmentation on LIP and COCO-stuff, VT-based feature pyramid networks (FPN) achieve 0.35 points higher mIoU while reducing the FPN module’s FLOPs by 6.5x.

1. Introduction

In computer vision, visual information is captured as arrays of pixels. These pixel arrays are then processed by convolutions, the de facto deep learning operator for computer vision. Although this convention has produced highly successful vision models, there are critical challenges:

1) Not all pixels are created equal: Image classification models should prioritize foreground objects over the background. Segmentation models should prioritize pedestrians over disproportionately large swaths of sky, road, vegetation etc. Nevertheless, convolutions uniformly process all image patches regardless of importance. This leads to spatial inefficiency in both computation and representation.

2) Not all images have all concepts: Low-level features such as corners and edges exist in all natural images, so applying low-level convolutional filters to all images is appropriate. However, high-level features such as ear shape exist in specific images, so applying high-level filters to all images is computationally inefficient. For example, dog features may not appear in images of flowers, vehicles, aquatic animals etc. This results in rarely-used, inapplicable filters expending a significant amount of compute.

3) Convolutions struggle to relate spatially-distant concepts: Each convolutional filter is constrained to operate on a small region, but long-range interactions between semantic concepts is vital. To relate spatially-distant concepts, previous approaches increase kernel sizes, increase model depth, or adopt new operations like dilated convolutions, global pooling, and non-local attention layers. However, by working within the pixel-convolution paradigm, these approaches at best mitigate the problem, compensating for the convolution by adding model complexity.

To overcome the above challenges, we address the root cause and introduce the Visual Transformer (VT) (Figure 1) to represent and process high-level concepts in images. Our intuition is that a sentence with a few words (or visual tokens) suffices to describe high-level concepts in a late-stage feature map. This motivates a departure from the fixed pixel-array representation later in the network; instead, we use spatial attention to convert the feature map into a compact set of semantic tokens. We then feed these tokens to a self-attention module or transformer [39] to capture token interactions. The resulting visual tokens computed can be directly used for image-level prediction tasks (e.g., classification) or be spatially re-projected to the feature map for pixel-level prediction tasks (e.g., segmentation). Unlike late-stage convolutions, our VT addresses the three challenges: 1) judiciously allocating computation by attending to important regions, instead of treating all pixels equally; 2) encoding semantic concepts in a few visual to-
Figure 1: Diagram of a Visual Transformer (VT). For a given image, we first apply convolutional layers to extract low-level features. The output feature map is then fed to VT: First, apply a tokenizer, grouping pixels into a small number of visual tokens, each representing a semantic concept in the image. Second, apply transformers to model relationships between tokens. Third, visual tokens are directly used for image classification or projected back to the feature map for semantic segmentation.

kens relevant to the image, instead of modeling all concepts across all images; and 3) relating spatially-distant concepts through self-attention in token-space.

To validate the effectiveness of VT and understanding its key components, we run controlled experiments by using VTs to replace convolutions in ResNet, a common test bed for new building blocks for image classification. We also use VTs to re-design feature-pyramid networks (FPN), a strong baseline for semantic segmentation. Our experiments show that VTs achieve higher accuracy with lower computational cost in both tasks. For the ImageNet[11] benchmark, we replace the last stage of ResNet[14] with VTs, reducing FLOPs of the stage by 6.9x and improving top-1 accuracy by 4.6 to 7 points.

Graph convolutions in vision models: Our work is also related to previous efforts such as GloRe [6], Latent-GNN [51], and [26] that densely relate concepts in latent space using graph convolutions. To augment convolutions, [26, 6, 51] adopt a procedure similar to ours: (1) extracting latent variables as graph nodes (analogous to our visual tokens) (2) applying graph convolution to capture node interactions (analogous to our transformer), and (3) projecting the nodes back to the feature map. Although these approaches avoid spatial redundancy, they are susceptible to concept redundancy: the second limitation listed in the introduction. In particular, by using fixed weights that are not content-aware, the graph convolution expects a fixed semantic concept in each node, regardless of whether the concept exists in the image. By contrast, a transformer uses content-aware weights, allowing visual tokens to represent varying concepts. As a result, while graph convolutions require hundreds of nodes (128 nodes in [4], 340 in [25], 150 in [52]) to encode potential semantic concepts, our VT uses just 16 visual tokens and attains higher accuracy. Furthermore, while [26, 6, 51] augment convolutions in a pretrained network, VTs replace convolutional layers to save FLOPs and parameters, and support training from scratch.

Attention in vision models: Attention is also widely used in different computer vision models [21, 20, 43, 46, 48, 42, 28, 18, 19, 1, 53, 30, 49]. Attention was first computed from the input and multiplied with the feature map [43, 21, 20, 46]. Later work [48, 34, 41] interprets this as a way to make convolutions spatially adaptive and content-aware. In [42], Wang et al. introduced non-local operators,
equivalent to self-attention, to video understanding to capture long-range interactions. However, self-attention is expensive, so [1] use self-attention in convolutions with small channel sizes and [30, 28, 7, 53, 19] restrict the receptive field of self-attention. Starting from [30], self-attention is used as a stand-alone building block for vision models. Our work is different from all above since we propose a novel token-transformer paradigm to replace the inefficient pixel-convolution paradigm and achieve superior performance.

**Efficient vision models:** Many works achieve better performance with lower computational cost. Early work in this direction includes [23, 31, 13, 17, 32, 16, 52, 27, 45]. Recent works use neural architecture search [44, 10, 40, 9, 37, 36] to automatically arrange existing convolution operators, which we show can be inefficient when used exclusively.

### 3. Visual Transformer

We illustrate the overall diagram of a *Visual Transformer* (VT) based model in Figure 1. First, process the input image with several convolution blocks, then feed the output feature map to VTs. Our insight is to leverage the strengths of both convolutions and VTs: (1) early in the network, use convolutions to learn densely-distributed, low-level patterns and (2) later in the network, use VTs to learn and relate more sparsely-distributed, higher-order semantic concepts. Use visual tokens for image-level prediction tasks and use the augmented feature map for pixel-level prediction tasks.

A VT module involves three steps: First, group pixels into semantic concepts, to produce a compact set of visual tokens. Second, to model relationships between semantic concepts, apply a transformer [39] to these visual tokens. Third, project these visual tokens back to pixel-space to obtain a new augmented feature map. With only 16 visual tokens, our VT outperforms previous methods [6, 51, 26] which use hundreds of semantic concepts (“nodes”).

#### 3.1. Tokenizer

Our intuition is that an image can be summarized by a few handfuls of words, or visual tokens. This contrasts convolutions, which use hundreds of filters, and graph convolutions, which use hundreds of “latent nodes” to detect all possible concepts regardless of image content. To leverage this intuition, we introduce a tokenizer module to convert feature maps into compact sets of visual tokens. Formally, we define the input feature map by $X \in \mathbb{R}^{HW \times C}$ (height $H$, width $W$, channels $C$) and visual tokens by $T \in \mathbb{R}^{L \times C}$ s.t. $L \ll HW$ ($L$ represents the number of tokens).

**3.1.1 Filter-based Tokenizer**

A filter-based tokenizer, also adopted by [51, 6, 26], utilizes convolutions to extract visual tokens. For feature map $X$, we map each pixel $X_p \in \mathbb{R}^C$ to one of $L$ semantic groups using point-wise convolutions. Then, within each group, we spatially pool pixels to obtain tokens $T$. Formally,

$$T = \text{SOFTMAX}_{HW}(XW_A)^T X$$

Here, $W_A \in \mathbb{R}^{C \times L}$ forms semantic groups from $X$, and SOFTMAX$_{HW}(\cdot)$ translates these activations into a spatial attention. Finally, $A$ multiplies with $X$ and computes weighted averages of pixels in $X$ to make $L$ visual tokens.

However, many high-level semantic concepts are sparse and may each appear in only a few images. As a result, the fixed set of learned weights $W_A$ potentially wastes computation by modeling all such high-level concepts at once. We call this a “filter-based” tokenizer, since it uses convolutional filters $W_A$ to extract visual tokens.

![Figure 2: Filter-based tokenizer that use convolution to group pixels using a fixed convolution filter.](image)

**3.1.2 Recurrent Tokenizer**

To remedy the limitation of filter-based tokenizers, we propose a recurrent tokenizer with weights that are dependent on previous layer’s visual tokens. The intuition is to let the previous layer’s tokens $T_{in}$ guide the extraction of new tokens for the current layer. The name of “recurrent tokenizer” comes from that current tokens are computed dependent on previous ones. Formally, we define

$$W_R = T_{in}W_{T \rightarrow R}, \quad T = \text{SOFTMAX}_{HW}(XW_R)^T X,$$

where $W_{T \rightarrow R} \in \mathbb{R}^{C \times C}$. This way VT can incrementally refine the set of tokens conditioned on previously-processed concepts. We apply recurrent tokenizers starting from the second VT, since it requires tokens from a previous VT.

#### 3.2. Transformer

After tokenization, we then need to model interactions between these visual tokens. Previous works [6, 51, 26] use graph convolutions to relate concepts. However, these operations use fixed weights during inference, meaning each token (or “node”) is bound to a specific concept, therefore graph convolutions waste computation by modeling all
high-level concepts, even those that only appear in few images. To address this, we adopt transformers [39], which use input-dependent weights by design. Due to this, transformers support visual tokens with variable meaning, covering more possible concepts with fewer tokens.

We employ a standard transformer with minor changes:

\[ \mathbf{T}_\text{out} = \mathbf{T}_\text{in} + \text{SOFTMAX}_L \left( (\mathbf{T}_\text{in} \mathbf{K})(\mathbf{T}_\text{in} \mathbf{Q})^T \right) \mathbf{T}_\text{in}, \]

where \( \mathbf{T}_\text{in}, \mathbf{T}_\text{out}, \mathbf{K}, \mathbf{Q} \in \mathbb{R}^{L \times C} \) are the visual tokens. Different from graph convolution, in a transformer, weights between tokens are input-dependent and computed as a key-query product: \( (\mathbf{T}_\text{in} \mathbf{K})(\mathbf{T}_\text{in} \mathbf{Q})^T \in \mathbb{R}^{L \times L} \). This allows us to use as few as 16 visual tokens, in contrast to hundreds of analogous nodes for graph-convolution approaches [6, 51, 26]. After the self-attention, we use a non-linearity and two pointwise convolutions in Equation (4), where \( \mathbf{F}_1, \mathbf{F}_2 \in \mathbb{R}^{C \times C} \) are weights, \( \sigma(\cdot) \) is the ReLU function.

3.3. Projector

Many vision tasks require pixel-level details, but such details are not preserved in visual tokens. Therefore, we fuse the transformer’s output with the feature map to refine the feature map’s pixel-array representation as

\[ \mathbf{X}_\text{out} = \mathbf{X}_\text{in} + \text{SOFTMAX}_L \left( (\mathbf{X}_\text{in} \mathbf{W}_Q)(\mathbf{TW}_K)^T \right) \mathbf{T}_\text{in}, \]

where \( \mathbf{X}_\text{in}, \mathbf{X}_\text{out} \in \mathbb{R}^{H \times W \times C} \) are the input and output feature map, \( \mathbf{X}_\text{in} \mathbf{W}_Q \in \mathbb{R}^{H \times W \times C} \) is the query computed from the input feature map \( \mathbf{X}_\text{in} \), \( \mathbf{X}_\text{in} \mathbf{W}_Q \cdot \mathbf{p} \in \mathbb{R}^C \) encodes the information pixel-\( p \) requires from the visual tokens. \( \mathbf{TW}_K \in \mathbb{R}^{L \times C} \) is the key computed from the token \( \mathbf{T} \). \( \mathbf{TW}_K \cdot i \in \mathbb{R}^C \) represents the information the \( i \)-th token encodes. The key-query product determines how to project information encoded in visual tokens \( \mathbf{T} \) to the original feature map. \( \mathbf{W}_Q \in \mathbb{R}^{C \times C}, \mathbf{W}_K \in \mathbb{R}^{C \times C} \) are learnable weights used to compute queries and keys.

4. Using VT in vision models

In this section, we discuss how to use VTs as building blocks in vision models. We define three hyper-parameters for each VT: channel size of the feature map; channel size of the visual tokens; and the number of visual tokens.

**Image classification model:** Following convention in image classification, we use ResNet backbones [14] to build visual-transformer-ResNets (VT-ResNets) by replacing the last stage of convolutions with VTs. First, we replace ResNet-[18, 34, 50, 101]’s 2 basic blocks, 3 basic blocks, 3 bottleneck blocks, and 3 bottleneck blocks, respectively, with the same number of VT modules. Second, since ResNet-[18, 34, 50, 101] outputs \( 14^2 \times 256, 14^2 \times 256, 14^2 \times 1024, 14^2 \times 1024 \) feature maps after stage-4 (before stage-5 max pooling), we set VT’s channel size to 256, 256, 1024, 1024. We use 16 visual tokens for all modules. The tokens are directly fed to the classification head – a standard average pool and fully-connected layer. Each model is exhaustively described in Appendix A. We reduce the last stage’s FLOPs by up to 6.9x (Table 1).

<table>
<thead>
<tr>
<th>FLOPs</th>
<th>Total</th>
<th>Stage-5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stage-5</td>
<td>1.14x</td>
<td>1.20x</td>
</tr>
<tr>
<td>Total</td>
<td>2.4x</td>
<td>5.0x</td>
</tr>
<tr>
<td>R18</td>
<td>256, 256</td>
<td>1024, 1024</td>
</tr>
</tbody>
</table>

Table 1: FLOPs and parameter size reduction of VTs on ResNets by replacing the last stage of convolution modules with VT modules.

**Semantic segmentation:** With convolutions, a) computational complexity grows with resolution and b) long-range spatial interactions are difficult to capture. However, VTs a) operate on a minimal set of visual tokens regardless of resolution and b) can capture long-range spatial interactions easily in latent space. We integrate VTs with the commonly-used panoptic feature pyramid networks (FPN) [24]. Panoptic FPNs extract ResNet feature maps at multiple stages and resolutions, which are then fused to produce multi-scale, detail-preserving feature maps (Figure 4 left). However, they rely on convolutions with large channel sizes operating on high resolution feature maps. We replace FPN convolutions with VT modules, producing VT-FPN (Figure 4 right). From each resolution, we extract 8 visual tokens with 1024 channels each, then relate tokens with a transformer before re-projecting back to the feature maps for pixel-level predictions. VT-FPN uses 6.4x fewer FLOPs than FPN with the same or better accuracy (Table 9 and 10).

5. Experiments

We conduct experiments with VTs on image classification and semantic segmentation to (a) understand the key components of VTs and (b) validate their effectiveness.

5.1. VT for Classification and Ablations

We experiment on ImageNet [11], which features 1.3 million training images and 50 thousand validating images.
Figure 4: Feature Pyramid Networks (FPN) (left) vs visual-transformer-FPN (VT-FPN) (right) for semantic segmentation. FPN uses convolution and interpolation to merge feature maps with different resolutions. VT-FPN extract visual tokens from all feature maps, merge them with one transformer, and project back to the original feature maps.

Table 2: VT-ResNet vs. baseline ResNets on the ImageNet dataset. By replacing the last stage of ResNets, VT-ResNet uses 224M, 384M fewer FLOPs than the baseline ResNets while achieving 1.7 points and 2.2 points higher validation accuracy. Note the training accuracy of VT-ResNets are much higher. This indicates VT-ResNets have higher model capacity and require stronger regularization (e.g., data augmentation) to fully utilize the model. See Table 8.

We implement VT models in PyTorch [29]. We use SGD with momentum [35]—an initial learning rate 0.1 decayed by 10x every 30 epochs, momentum 0.9, weight decay $4\times 10^{-5}$, batch size 256, and 90 epochs. We use 8 V100 GPUs.

**VT vs. ResNet with default training recipe:** In Table 2, we compare VT-ResNets and vanilla ResNets under the same training recipe. VT-ResNets replace the last stage with a string of VT modules, using a filter-based tokenizer for the first module and recurrent tokenizers for subsequent modules. VT-ResNets outperform baselines by up to 2.1 points despite using fewer FLOPs—224M fewer (VT-R18) and 384M fewer (VT-R34). Furthermore, VT-ResNets overfit more heavily, with 7.9 (VT-R18) and 6.9 (VT-R34) points higher training accuracy. We hypothesize this is because VT-ResNets have much larger capacity and we need stronger regularization (e.g., data augmentation). We address this in Section 5.2 and Table 8.

**Tokenizer ablation:** We replace the first VT module’s tokenizer with simpler baselines: First, we consider a naive pooling-based tokenizer, which simply bilinearly interpolates a feature map spatially, to reduce from $HW = 196$ to $L = 16$. Second, we consider a clustering-based tokenizer (Appendix C), which clusters pixels using k-means to form tokens. Per Table 3, the naive pooling-based tokenizer underperforms by a significant margin, validating the efficacy of “smarter” pixel grouping. However, filter-based and clustering-based tokenizers perform similarly, with opposite rankings between VT-R18 and VT-R34. We hypothesize this is due to complementary drawbacks: Filter-based tokenizers are limited by fixed convolutional filters with non-content-aware weights, and clustering-based tokenizers extracts concepts that may not be critical for downstream classification performance. In Table 4, we validate the recurrent tokenizer’s effectiveness.

**Modeling token relationships:** In Table 5, we compare different methods of capturing token relationships. Both (a) the baseline without computing token interactions and (b) graph convolutions graph convolutions [6, 26, 51] underperform VTs, validating the need for both token interaction and for content-aware token extraction.

**Token efficiency ablation:** In Table 6, we test varying
Chapter 5: VT-ResNets using different modules to model token relationships. Models using transformers perform better than graph-convolution or no token-space operations. This validates that it is important to model relationships between visual token (semantic concepts) and transformer work better than graph convolution in relating tokens.

Table 5: VT-ResNets using different modules to model token relationships. Models using transformers perform better than graph-convolution or no token-space operations. This validates that it is important to model relationships between visual token (semantic concepts) and transformer work better than graph convolution in relating tokens.

Table 6: Using more visual tokens do not improve the accuracy of VT by significant margins, which agrees with our hypothesis that images can be described by a compact set of visual tokens.

Table 7: VTs that projects tokens back to feature maps perform better. This may be because feature maps still encode important spatial information.

5.2. Training VT with Advanced Recipe

In Table 2, we show that under the regular training recipe, the VT-ResNets experience serious overfitting, with higher validation accuracy but even larger train-val accuracy gap than the baseline. We thus hypothesize VT-based models have much higher model capacity. To maximize this, we retrain with advanced training recipes, using more training epochs, stronger data augmentation, stronger regularization, and distillation. Specifically, we use 400 epochs, RMSProp, initial learning rate 0.01, 5 warmup epochs increasing learning rate to 0.16, then a learning rate reduction of 0.9875 per epoch, synchronized batch normalization, distributed training with batch size 2048, label smoothing, AutoAugment [8], stochastic depth survival probability [22] 0.9, dropout ratio 0.2, exponential moving average (EMA) with 0.99985 decay, and knowledge distillation [15] with FBNetV3-G [9] as teacher. The final loss weights the distillation term by 0.8 and cross entropy term by 0.2.

Our results are reported in Table 8. Compared with the baseline ResNet models, VT-ResNet models achieve 4.6 to 7 points higher accuracy. Our VT-ResNets furthermore outperform other ResNet-based attention variants [21, 43, 1, 5, 19, 30, 53, 6]. This validates that our advanced training recipe better utilizes the VT-ResNet’s model capacity. We also compare with concurrent work that adopt transformers in vision models [12, 38, 50, 33] though our work is earlier than these papers. Our models outperform competitors despite using far fewer FLOPs and parameters.

Each of the included baselines utilizes their own training recipes, in addition to their architectural changes; to understand the source of our accuracy gain, we train ResNet18 and ResNet34 with the same advanced training recipe. Despite this, the accuracy gap between VT-ResNet and ResNets increases from 1.7 and 2.2 points to 2.2 and 3.0 points, respectively, despite using fewer FLOPs and parameters. This further validates that a stronger training recipe can better utilize VT model capacity. For this last stage, we observe FLOP reductions of up to 6.9x (Table 1).

5.3. Visual Transformer for Semantic Segmentation

We conduct experiments to test the effectiveness of VT for semantic segmentation on the COCO-stuff [2] and LIP [25] datasets. The COCO-stuff dataset contains annotations for 91 stuff classes with 118K training images and 5K validation images. LIP dataset is human image dataset with challenging poses and views. For the COCO-stuff dataset, we train a VT-FPN model with ResNet-\{50, 101\} backbones. Our implementation is based on Detectron2 [47]. Our training recipe is based on the semantic segmentation FPN recipe with 1x training steps, except that we use synchronized batch normalization in the VT-FPN, change the batch size to 32, and use a base learning rate of 0.04. For the LIP dataset, we also use synchronized batch normalization with a batch size of 96. We train the model with SGD using weight decay of 0.0005 and learning rate of 0.01.

As we can see in Table 9 and 10, after replacing FPN with VT-FPN, both ResNet-50 and ResNet-101 based models achieve slightly higher mIoU, but VT-FPN requires 6.5x fewer FLOPs than a FPN module.
cient, suffering from accuracy loss when compared with concurrent work, ViT [12] replaces convolutions at all stages of the network with transformers. However, we find using transformers early in a network is extremely inefficient, suffering from accuracy loss when compared with baselines with similar resource constraints (Table 8). To study why, we analyze a pre-trained ViT-B/16 and find its self-attention patterns in a pretrained ViT-B/16 model. 

### 6. Analyses

**Why not use transformers at early stages?** One prominent concurrent work, ViT [12] replaces convolutions at all stages of the network with transformers. However, we find using transformers early in a network is extremely inefficient, suffering from accuracy loss when compared with

<table>
<thead>
<tr>
<th>Models</th>
<th>Top-1 Acc (%)</th>
<th>FLOPs (G)</th>
<th>Params (M)</th>
</tr>
</thead>
<tbody>
<tr>
<td>R18[14]</td>
<td>69.8</td>
<td>1.814</td>
<td>11.7</td>
</tr>
<tr>
<td>R18+SE[21, 43]</td>
<td>70.6</td>
<td>1.814</td>
<td>11.8</td>
</tr>
<tr>
<td>R18+CBAM[43]</td>
<td>70.7</td>
<td>1.815</td>
<td>11.8</td>
</tr>
<tr>
<td>LR-R18[19]</td>
<td>74.6</td>
<td>2.5</td>
<td>14.4</td>
</tr>
<tr>
<td>R18<a href="ours">14</a></td>
<td>73.8</td>
<td>1.814</td>
<td>11.7</td>
</tr>
<tr>
<td>VT-R18(ours)</td>
<td>76.8</td>
<td>1.569</td>
<td>11.7</td>
</tr>
<tr>
<td>R34[14]</td>
<td>73.3</td>
<td>3.664</td>
<td>21.8</td>
</tr>
<tr>
<td>R34+SE[21, 43]</td>
<td>73.9</td>
<td>3.664</td>
<td>22.0</td>
</tr>
<tr>
<td>R34+CBAM[43]</td>
<td>74.0</td>
<td>3.664</td>
<td>22.9</td>
</tr>
<tr>
<td>AA-R34[1]</td>
<td>74.7</td>
<td>3.55</td>
<td>20.7</td>
</tr>
<tr>
<td>R34<a href="ours">14</a></td>
<td>77.7</td>
<td>3.664</td>
<td>21.8</td>
</tr>
<tr>
<td>VT-R34(ours)</td>
<td>79.9</td>
<td>3.236</td>
<td>19.2</td>
</tr>
<tr>
<td>R50[14]</td>
<td>76.0</td>
<td>4.089</td>
<td>25.5</td>
</tr>
<tr>
<td>R50+SE[21, 43]</td>
<td>76.9</td>
<td>3.860*</td>
<td>28.1</td>
</tr>
<tr>
<td>R50+CBAM[43]</td>
<td>77.3</td>
<td>3.864*</td>
<td>28.1</td>
</tr>
<tr>
<td>LR-R50[19]</td>
<td>77.3</td>
<td>4.3</td>
<td>23.3</td>
</tr>
<tr>
<td>Stand-Alone[30]</td>
<td>77.6</td>
<td>3.6</td>
<td>18.0</td>
</tr>
<tr>
<td>AA-R50[1]</td>
<td>77.7</td>
<td>4.1</td>
<td>25.6</td>
</tr>
<tr>
<td>A²-R50[5]</td>
<td>77.0</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>SAN19[53]</td>
<td>78.2</td>
<td>3.3</td>
<td>20.5</td>
</tr>
<tr>
<td>GloRe-R50[6]</td>
<td>78.4</td>
<td>5.2</td>
<td>30.5</td>
</tr>
<tr>
<td>VT-R50(ours)</td>
<td>80.6</td>
<td>3.412</td>
<td>21.4</td>
</tr>
<tr>
<td>R101[14]</td>
<td>77.4</td>
<td>7.802</td>
<td>44.4</td>
</tr>
<tr>
<td>R101+SE[21, 43]</td>
<td>77.7</td>
<td>7.575*</td>
<td>49.3</td>
</tr>
<tr>
<td>R101+CBAM[43]</td>
<td>78.5</td>
<td>7.581*</td>
<td>49.3</td>
</tr>
<tr>
<td>LR-R101[19]</td>
<td>78.5</td>
<td>7.79</td>
<td>42.0</td>
</tr>
<tr>
<td>AA-R101[1]</td>
<td>78.7</td>
<td>8.05</td>
<td>45.4</td>
</tr>
<tr>
<td>GloRe-R200[6]</td>
<td>79.9</td>
<td>16.9</td>
<td>70.6</td>
</tr>
<tr>
<td>VT-R101(ours)</td>
<td>82.3</td>
<td>7.129</td>
<td>41.5</td>
</tr>
<tr>
<td>VT-B/16-224 [12]†</td>
<td>79.7</td>
<td>17.61</td>
<td>86.4</td>
</tr>
<tr>
<td>DetF/B-16-224[38]</td>
<td>81.8</td>
<td>17.61</td>
<td>86</td>
</tr>
<tr>
<td>T2T-ViT-224 [50]</td>
<td>82.2</td>
<td>13.2</td>
<td>64.1</td>
</tr>
<tr>
<td>BoTNet-S159 [33]</td>
<td>81.7</td>
<td>7.3</td>
<td>33.5</td>
</tr>
</tbody>
</table>

Table 8: Comparing VT-ResNets with other attention-augmented ResNets on ImageNet. *The baseline ResNet FLOPs reported in [43] is lower than our baseline. † We are citing the accuracy of training from scratch at a resolution of 224 from [50]. ‡ FLOP estimation is cited from [50]. Figure 9 in the Appendix shows a plot of accuracy vs. parameters and FLOPs for models above.

<table>
<thead>
<tr>
<th></th>
<th>mIoU (%)</th>
<th>Total FLOPs (G)</th>
<th>FLOPs (G)</th>
</tr>
</thead>
<tbody>
<tr>
<td>R50 FPN</td>
<td>47.04</td>
<td>37.1</td>
<td>12.8</td>
</tr>
<tr>
<td>VT-FPN</td>
<td>47.39</td>
<td>26.4 (1.41x)</td>
<td>2.0 (6.40x)</td>
</tr>
<tr>
<td>R101 FPN</td>
<td>47.35</td>
<td>54.4</td>
<td>12.8</td>
</tr>
<tr>
<td>VT-FPN</td>
<td>47.58</td>
<td>43.6 (1.25x)</td>
<td>2.0 (6.40x)</td>
</tr>
</tbody>
</table>

Table 10: Semantic segmentation results on the Look Into Person validation set. The FLOPs are calculated with a typical input resolution of 473×473.

![Figure 5](image_url)

Figure 5: The upper image presents the percentage of local attention over the all attention values at different layers. The solid line is the mean of all the heads and the boundaries denote the standard deviations. The bottom row are self-attention patterns in a pretrained ViT-B/16 model.
Figure 6: Visualization of the spatial attention generated by a filter-based tokenizer on images from the LIP dataset. Red denotes higher attention values and color blue denotes lower. Without any supervision, visual tokens automatically focus on different areas of the image that correspond to different semantic concepts, such as sheep, ground, clothes, woods. Row 1 shows pixels contributions to each token, and Row 2 shows how different pixels interact with the same token. Note in Row 2 that pixels may be from disparate, spatially-distant portions of the image, indicating VT can capture long-range interactions.

7. Conclusion

A recent trend in computer vision replaces convolutions with transformers. However, this ignores the motivation for a convolution: convolutions are efficient for processing highly-redundant, highly-localized patterns like edges and corners, which occur early in a network. In lieu of this, we design convolution-transformer hybrids that leverage the strengths of both operations. We propose Visual Transformers (VTs), learning and relating sparsely-distributed, high-level concepts far more efficiently: Instead of pixel arrays, VTs represent just the high-level concepts in an image using visual tokens. Instead of convolutions, VTs apply transformers to directly relate semantic concepts in token-space. To evaluate this idea, we replace convolutional modules with VTs, obtaining significant accuracy improvements across tasks and datasets. Using an advanced training recipe, our VT improves ResNet accuracy on ImageNet by 4.6 to 7 points. For semantic segmentation on LIP and COCO-stuff, VT-based feature pyramid networks (FPN) achieve 0.35 points higher mIoU despite 6.5x fewer FLOPs than convolutional FPN modules. This paradigm can furthermore be compounded with other contemporaneous tricks beyond the scope of this paper, including extra training data and neural architecture search. However, instead of presenting a mosh pit of deep learning tricks, our goal is to show that the pixel-convolution paradigm is fraught with redundancies, which can be mitigated by tackling the root cause – addressing redundancy in the pixel-convolution convention by adopting the token-transformer paradigm, instead of exacerbating compute demands.

What do the tokens learn? We show that extracted VT tokens correspond to different semantic image regions, by visualizing the spatial attention $A \in \mathbb{R}^{HW \times L}$ for filter-based tokenizers in Figure 6, Row 1. Attention maps $A_{i,l} \in \mathbb{R}^{HW}$ reflect each pixel’s contribution to token-$l$, showing how each token represents different semantic parts of the scene. We also find VT trained on LIP assigns 28.3% higher attention to foreground pixels (annotated parts) than to background pixels. See more visual cases in Appendix B.

Does VT treat each pixel equally? We find, as hypothesized, VT assigns computation non-uniformly spatially. This is verified by the non-uniform attention distribution across the image, as shown in Figure 6, Row 1. We also quantify this by computing the visualized attention map $A$’s entropy $E = - \sum_{i,j} A_{i,j} \log(A_{i,j})$. For a convolution, $\forall i,j, A_{i,j} = 1/(HW)$. We use $473 \times 473$ for LIP images, making the baseline entropy $E_{conv} = 12.318$. For VT, the attention is $A \in \mathbb{R}^{HW \times L}$ (Section 3.1.1), making VT entropy $E_{vt} = 0.941$. This is 13x smaller than $E_{conv}$, verifying VT does not treat each pixel equally.

Does VT capture long-range interactions? We design VT hoping it can capture long-range interactions and overcome the limitation of convolutions. We verify this by analyzing which pixels are interacting with each token. Formally, for token-$l$ with attention map $A_{i,j,l}$, its interaction with other tokens is captured by the self-attention weight $W_{i,l}$ computed in Equation (3) We analyze which pixels interacted with token-$l$ by computing $A_{i,j,l} = \sum_{l'} W_{i,l'} \times A_{i,j,l'}$. We visualize $A_{i,j,l}$ in Fig. 6 Row 2. Same as $A$, $A$ attends globally to the entire image. The focus regions of $A$ can be disparate, spatially distant portions of the image than $A$, indicating VTs capture long-range interactions.
References


[26] Xiaodian Liang, Zhitong Hu, Hao Zhang, Liang Lin, and Eric P Xing. Symbolic graph reasoning meets convolu-