This CVPR workshop paper is the Open Access version, provided by the Computer Vision Foundation.
Except for this watermark, it is identical to the accepted version;
the final published version of the proceedings is available on IEEE Xplore.

Vision Transformers with Mixed-Resolution Tokenization

Tomer Ronen
Tel Aviv University
tomer.ronen34@gmail.com

Omer Levy
Tel Aviv University

Avram Golbert
Google Research*

Abstract

Vision Transformer models process input images by dividing them into a spatially regular grid of equal-size patches. Conversely, Transformers were originally introduced over natural language sequences, where each token represents a subword – a chunk of raw data of arbitrary size. In this work, we apply this approach to Vision Transformers by introducing a novel image tokenization scheme, replacing the standard uniform grid with a mixed-resolution sequence of tokens, where each token represents a patch of arbitrary size. Using the Quadtree algorithm and a novel saliency scorer, we construct a patch mosaic where low-saliency areas of the image are processed in low resolution, routing more of the model's capacity to important image regions. Using the same architecture as vanilla ViTs, our Quadformer models achieve substantial accuracy gains on image classification when controlling for the computational budget. Code and models are publicly available at https://github.com/TomerRonen34/mixed-resolution-vit.

1. Introduction

Transformer [37] models are designed to process sequential input data. Vision Transformer (ViT) [6] models process input images that naturally have two spatial dimensions, requiring a spatially-aware tokenization scheme to convert them into sequences. The vast majority of Vision Transformers convert the input image into a two-dimensional grid of token vectors, before flattening it to create a one-dimensional sequence. Specifically, most methods use uniform patch tokenization, splitting the image into a spatially regular grid of equal-size patches.

In natural language processing, input tokenization looks entirely different. Almost all modern neural networks for text processing use subword tokenization, where each token represents a substring of arbitrary character length [15, 31]. In this work, we apply this approach to ViTs by introducing a novel image tokenization scheme, replacing the standard uniform grid with a mixed-resolution sequence of tokens, where each token represents a patch of arbitrary size.

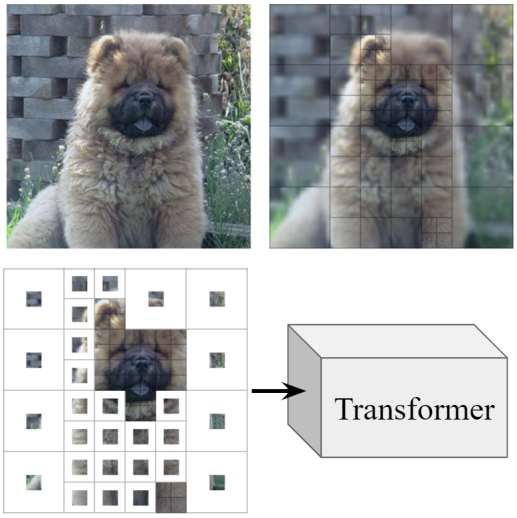

Figure 1. The Quadformer. We split the image into a mixed-resolution patch mosaic according to a saliency scorer, and employ a standard Transformer architecture with 2D position embeddings.

Previous works tried to incorporate multi-resolution processing into Vision Transformers by building feature pyramids inspired by the structure of CNNs [10, 39], using multi-resolution attention [33, 44], or merging intermediate token representations from across the entire image without preserving spatial locality [2, 26]. In contrast, our work is the first to use mixed-resolution *tokenization*, directly splitting the input image into a patch mosaic processed by a standard Transformer model (see Figure 1).

Instead of using a spatially regular patch grid, we construct a patch mosaic where low-saliency areas of the image are processed in low resolution, routing more of the model's capacity to important areas. Practically, we use the Quadtree algorithm [19] to recursively split the image into patches of different sizes, incorporating a saliency scorer that chooses which areas of the image to split by their estimated importance. We use 2D position embeddings to represent the location of each patch.

*The author was affiliated with Alibaba Group during parts of the research.

We evaluate our method, dubbed *Quadformer*, on the ImageNet-1k [28] classification dataset, and compare our mixed-resolution models to vanilla ViT models that use the same architecture. While vanilla ViT models utilize uniform grid tokenizations with a single patch size (in our case, the standard 16^2 pixels), our mixed-resolution tokenization uses 3 patch sizes (64^2, 32^2 and 16^2 pixels), allowing our Quadformer models to process important image regions in high resolution even when using a small number of patches. Using a novel saliency scorer based on neural representations, we consistently beat the accuracy of vanilla ViTs by up to 0.88 absolute percentage points when controlling for the number of patches or GMACs. Despite not using dedicated tools for accelerated inference, we also show gains when controlling for inference speed, beating vanilla ViT models by up to 0.42 absolute percentage points.

2. Background and related work

Efficient Vision Transformers. Many efficient architectures were proposed for improving the speed-accuracy tradeoff of Vision Transformers, mostly by using attention layers with linear time complexity [1, 16, 36], dropping a subset of patches [21,25,45], or merging intermediate token representations from the entire image [2, 26]. Our method offers orthogonal improvements as we decrease the number of patches via tokenization, maintaining global attention over the entire image while using spatially-local tokens.

Vision Transformers with spatially uniform grids. Standard Vision Transformer models process input images by dividing them into a regular grid of equal size patches. Even in the case of pyramid vision transformers [16, 39], which gradually compress the spatial dimension of the feature map as the network progresses, vectors in the same feature map always represent input areas of the same size. This is a classical design choice used extensively with CNNs, as it fits the constraints of convolution layers, that must operate on a spatially-regular grid. However, the layers that form the Transformer model, namely self-attention layers and fully connected layers, have no such limitations. Transformer models can process any set of input vectors that have some defined positional relationship, and are naturally suited to handling inputs of different scales. For example, Transformer language models process input tokens that represent subwords of very different lengths – the BERT [5] vocabulary has tokens in lengths ranging from 1 character ("a", "b") to 18 characters ("telecommunications").

Existing methods for image tokenization. Not all Vision Transformers use the standard uniform grid tokenization scheme. Some methods use CNN backbones to create representations from input images, using the activation volumes as tokens [10,42]. Another class of Vision Transformers designed for image generation uses vector-quantization networks to learn a codebook of discrete tokens, also using a uniform two-dimensional grid [7, 24]. Few methods forgo spatial tokenization altogether and employ a technique called token learning, where each token aggregates information from the entire image [29].

Figure 2. Tokenizations obtained using our saliency-based Quadtree. For clearer visualization, we upsample patches back to their original size after the tokenizer resizes them to a fixed representation size. Notice how high-saliency regions are represented in high resolution while background regions are blurry.

Quadtrees. Quadtrees are data structures that recursively split a two-dimensional space into a tree of quadrants, where each internal node has exactly four children. Each node in the tree represents a specific spatial area defined by an axis-aligned rectangle or square. Leaf nodes store the information contained in the area they represent. Quadtrees were originally developed for fast retrieval of 2D points [8]. They were quickly adapted for image analysis [11], and later for image compression [19].

Quadtrees and neural networks. Few successful attempts have been made to integrate the Quadtree algorithm with neural networks. To the best of our knowledge, our work is the first to use Quadtree representations of RGB images as inputs to a neural net.

Jewsbury *et al*. [13] use Quadtrees to divide large pathology images into smaller subimages, with each subimage individually processed by a standard CNN. Many works on 3D shape analysis [27, 34, 38] use specialized CNN architectures to process Octrees [20], the 3D equivalent of Quadtrees. Jayaraman *et al*. [12] use Quadtrees with sparse CNNs to process simple black-and-white sketches, avoiding computation in blank areas of the image. Chitta *et al*. [3] use Quadtrees with a sparse CNN decoder to predict hierarchical segmentation maps, avoiding excessive computation

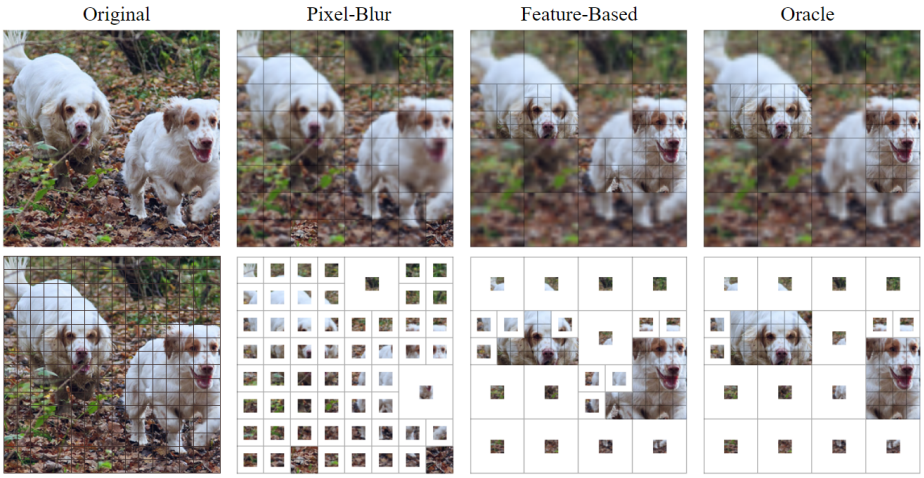

Figure 3. The effect of different patch scorers on Quadtree tokenization. Better saliency estimator → higher resolution in important areas. The pixel-blur scorer is often used for image compression, as it focuses on high-frequency details. Our feature-based scorer estimates patch saliency using neural representations. The oracle scorer uses the Grad-CAM saliency estimation algorithm.

in large image regions that share the same class. Tang *et al*. [33] propose an efficient attention implementation for ViTs, where each query vector in a spatially uniform grid attends to a Quadtree of key-value vectors. Ke *et al*. [14] use Quadtrees for efficient refinement of instance segmentation masks, focusing computation in incoherent regions. They employ a Transformer model over a Quadtree of feature vectors extracted from a CNN feature pyramid.

3. Method

3.1. ViTs with mixed-resolution tokenization

We define a **mixed-resolution patch mosaic** to be a division of an image into a set of non-overlapping patches of different sizes, such that the entire area of the image is covered (see examples in Figure 1, Figure 2, Figure 3). With small adaptions to the way ViT models represent image patches, we convert mixed-resolution patch mosaics into token sequences that can be processed by a standard Transformer model. These adaptations deal with 2 aspects of the tokens: patch embedding and position embedding.

Patch embedding: each patch in the mosaic is resized to a fixed representation size (e.g. 16^2 pixels), then flattened and passed through a shared fully connected layer. Notice that all patches are represented by tokens of equal dimension, regardless of the area they cover in the image.

Position embedding: the learned 1-dimensional position embeddings common in vanilla ViTs lose meaning when the patches are not part of a regular grid. Instead, we use 2-dimensional position embeddings. We embed the x and y positions separately, then concatenate them to create the final position embedding, as suggested by Dosovitskiy *et al*. [6] We use the (x, y) position of the center of the patch inside a grid determined by the smallest patch size.

Input:
Image $im \in \mathbb{R}^{h \times w \times 3}$,
desired number of patches $L \in \mathbb{N}$,
patch edge sizes $s_{min}, s_{max} \in \mathbb{N}$,
saliency scorer $score : patch \to \mathbb{R}^+$
Output:
The set of chosen patches P_{chosen}

Algorithm:
$P_{chosen} \leftarrow$ slice im into a uniform grid with patch size s_{max}
while $|P_{chosen}| < L$ **do**
  $P_{splittable} \leftarrow \{p \mid p \in P_{chosen} \ \& \ size(p) \geq 2s_{min}\}$
  $p_{split} \leftarrow \arg\max_{p \in P_{splittable}} score(p)$
  $children(p_{split}) \leftarrow$ divide p_{split} into 4 quadrants
  $P_{chosen} \leftarrow children(p_{split}) \cup P_{chosen} \setminus \{p_{split}\}$
end
Return P_{chosen}

Algorithm 1: The saliency-based Quadtree. We iteratively choose the "most important" image region as ranked by a saliency scorer and split it into 4 quadrants. In practice, we run the algorithm on a batch of images for improved speed, taking only $19\mu\text{-}secs$ per image for the splitting logic. Patch scoring is also batched, taking 19–$157\mu\text{-}secs$ per image depending on the scorer. See subsection 4.2 and Table 1 for more details.

3.2. Saliency-based Quadtrees

Quadtrees for RGB images. Quadtrees are data structures that recursively split a two-dimensional space into a tree of quadrants, where each internal node has exactly four children. Each node in the tree represents a specific spatial area defined by an axis-aligned rectangle or square. In Quadtrees that represent RGB images, each leaf contains a compressed representation of an image patch, often a copy of that patch downsampled to some predetermined size.

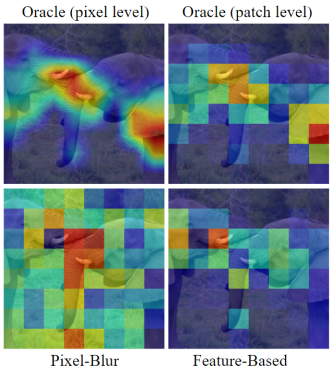

Figure 4. Patch saliency maps created by different scorers for an image labeled "African Elephant".

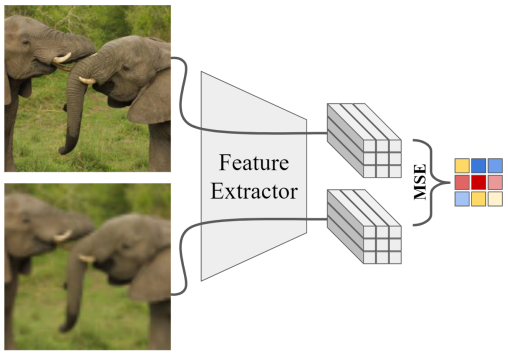

Figure 5. Feature-based patch scorer. The MSE between a patch representation and its blurry counterpart estimates the semantic information loss from decreasing the resolution of the patch.

Typically, Quadtrees for RGB images are constructed by a top-down algorithm (Algorithm 1), which iteratively chooses the "most important" image patch as ranked by a scoring function and splits it into 4 patches, effectively using 4 times more pixels to represent the selected image region. We call this scoring function a "patch scorer".

We use the Quadtree algorithm as a tokenizer, splitting images into mixed-resolution patch mosaics which we then feed into a standard Transformer model. We experiment with several patch scorers (Figure 3): the pixel-blur scorer commonly used for Quadtree image compression, a novel feature-based scorer that estimates saliency using neural representations, and a Grad-CAM oracle scorer which utilizes a label-aware saliency method and gives a loose upper bound on the scoring quality we can hope to achieve.

Pixel-blur scorer. In image compression applications, Quadtree patch scoring often relies on the MSE between an image patch and a compressed representation of that patch [19], such as a blurry version of the patch obtained by downsampling it to the Quadtree representation size and upsampling back to the original size. This score estimates the pixel-level information loss caused by decreasing the resolution of the patch. Let p be an image patch:

$$\begin{aligned} p_{blur} &= upsample\big(downsample(p)\big) \\ score_{PixelBlur}(p) &= MSE\big(p,\ p_{blur}\big) \end{aligned} \tag{1}$$

The pixel blur scorer assigns high importance to areas of the image with a lot of high-frequency content, since calculating the difference between a patch and its blurry counterpart is equivalent to running a high-pass filter. While high-frequency content may be a good importance measure for image compression, it is a poor measure of object saliency, as natural images often have detailed backgrounds or textures that are insignificant when trying to identify the objects in the image. To address this misalignment between the patch scorer objective and the model objective, we propose a different scorer based on semantic representations.

Feature-based scorer. Computer vision neural networks are often used to extract semantically meaningful feature vectors. Both Vision Transformers and CNNs create contextualized embeddings of image regions: ViTs have an explicit mapping between feature vectors to image patches, and CNNs create a spatially-aware convolutional activation volume for the entire image wherein each feature vector can be mapped implicitly to a corresponding image region.

Using these neural representations, we introduce $score_{Feat}$, a patch scorer that estimates the semantic information loss from decreasing the resolution of an image patch by comparing its original representation to its representation in a blurred image (Figure 5). Intuitively, this score estimates how much semantic information is lost when we downsample the patch from its original size to the Quadtree representation size. For example, if the Quadtree representation size is 16^2 pixels, the features of a 64^2 patch in full resolution are compared to the features of this patch when the image is blurred by a factor of $\frac{64}{16} = 4$.

Formally, let $im \in \mathbb{R}^{h_{im} \times w_{im} \times 3}$ and $blur(im, x)$ be an RGB image and its corresponding blurred image obtained by downsampling the image by a factor of x and upsampling it back to the original size. We extract a feature map $feat(im) \in \mathbb{R}^{H \times W \times d}$ by running a feature extractor NN on the image im. Given an image patch p of size $s_p \times s_p$, we slice the region in $feat(im)$ which corresponds to p's location in the image: $feat(im)[p] \in \mathbb{R}^{\frac{s_p}{h_{im}} H \times \frac{s_p}{w_{im}} W \times d}$. We use $feat(im)[p]$ as a semantic representation of p, a technique very similar to RoI pooling [9]. Given the Quadtree representation size $s_{rep} \in \mathbb{N}$, we use these notations to define $score_{Feat}$:

$$\begin{aligned} im_{blur} &= blur\big(im, \frac{s_p}{s_{rep}}\big) \\ score_{Feat}(p) &= MSE\big(feat(im_{blur})[p],\ feat(im)[p]\big) \end{aligned} \tag{2}$$

Grad-CAM oracle scorer. Grad-CAM [30] is a method for creating visual explanations of predictions made by a variety of computer vision models. For classification nets, given an image and a target class, Grad-CAM produces a pixel-level saliency map where the weight attributed to each pixel represents its importance in classifying the image to the given target class. Using average pooling, we turn this saliency map into patch scores suitable for the saliency-based Quadtree algorithm (Figure 4). To estimate a loose upper bound on the accuracy we can hope to achieve with Quadformer models, we use specific oracle Quadformers, which we train and evaluate with the high-quality saliency scores produced by a Grad-CAM patch scorer that is aware of the actual ground-truth label of the input images.

4. Experiments

4.1. Dataset and evaluation metrics

We conduct experiments on ImageNet-1K [28] and report the top-1 accuracy trade-off with respect to several cost indicators, as suggested by Dehghani *et al*. [4] To evaluate model efficiency, we report the number of patches/tokens in the input to the Transformer model, the number of giga multiply–accumulate operations (GMACs) per image as estimated by fvcore [43], and the throughput (ims/sec) and runtime (μ-secs/im) on a single GeForce RTX 3090 GPU, measured with timm [41] with batch size 512 in mixed precision. We do not use parameter count as a cost indicator since our Quadformer models use the exact same architectures as our vanilla ViT models: ViT-Small (22M params), ViT-Base (86M params) and ViT-Large (307M params). Some Quadformer models use a neural net for saliency estimation, but since it only has 342K parameters (see §4.2) its impact on the parameter count is negligible.

4.2. Implementation details

Base models. All our base models use image size 256^2, patch size 16^2, and $2D$ sinusoidal position embeddings. For our two main ViT architectures — ViT-Base and ViT-Large — we start by taking the weights released by the original authors [6], which are pretrained on ImageNet-21K and fine-tuned on ImageNet-1K. These pretrained models use learned 1D position embeddings, image size 224^2, and patch size 16^2. We adapt them to $2D$ sinusoidal position embeddings and image size 256^2 by fine-tuning on ImageNet-1K with base learning rate 1e-4 for 70 epochs (for ViT-Base) or 20 epochs (for ViT-Large). For each architecture, we choose the checkpoint that achieved the highest validation accuracy. For ViT-Small, we train the DeiT-S architecture [35] from scratch on ImageNet-1K with base learning rate 2e-3 for 310 epochs.

Fine-tuning. We use the base models to initialize the weights of all our fine-tuned models, as we have seen much faster conversion times compared to training from scratch. We use the same base models to initialize both vanilla Vision Transformers and Quadformer models, as Quadformers share the exact same architecture with vanilla ViTs and do not introduce any extra parameters, except those used in the tokenizer.

Our Quadformer models use mixed-resolution tokenizations with patch sizes 64^2, 32^2 and 16^2 pixels, all downsampled to a patch representation size of 16^2 pixels. We fix the image size to 256^2 pixels and control the number of patches by setting the number of splits done by the Quadtree algorithm. Our vanilla ViT models use patch size 16^2. We control the number of patches by setting the image size to $(16\sqrt{\#Patches})^2$ pixels. We report detailed hyperparameters in the supplementary material.

Patch scorers. For our feature-based patch scorer, we use a ShuffleNetV2$\times$0.5 [17] model trained on ImageNet-1K as the feature extractor. We truncate it just before the fully connected classification layer, which results in a $\times$32 downscaling ratio. This feature extraction backbone has only 342K parameters, which adds little overhead and makes it practical for real-world inference purposes. For Quadformer models that use ViT-Base or ViT-Small, we perform the scoring on a $\times$0.75 downsampled image (with 192^2 pixels) and then upsample the saliency map by the same ratio, since the increased speed compensates for the lower fidelity and results in a better speed-accuracy tradeoff.

For Grad-CAM oracle saliency estimation we use a RegNetY-32GF [23] model with 145M parameters, learned via transfer learning by end-to-end fine-tuning the original SWAG [32] weights on ImageNet-1K data. Weights for ShuffleNetV2$\times$0.5 and RegNetY-32GF are taken from the torchvision library [18].

Quadtree. We build our own PyTorch [22] implementation of the Quadtree algorithm (Algorithm 1), using z-order curves for efficient tree construction [40]. Patch scores are computed for an entire batch of input images over all possible spatial locations. Since we use a top-down algorithm, the only valid candidates are subdivisions of the initial patch grid, resulting in a total of 80 splittable patches for images of size 256^2 pixels. All our patch scorers employ image-wide computation followed by grid-based scoring, making them particularly suitable for this kind of batched computation. The argmax operation used for iterative splitting is also batched, as well as the image slicing and resizing required to create token representations, making the entire implementation very GPU-friendly.

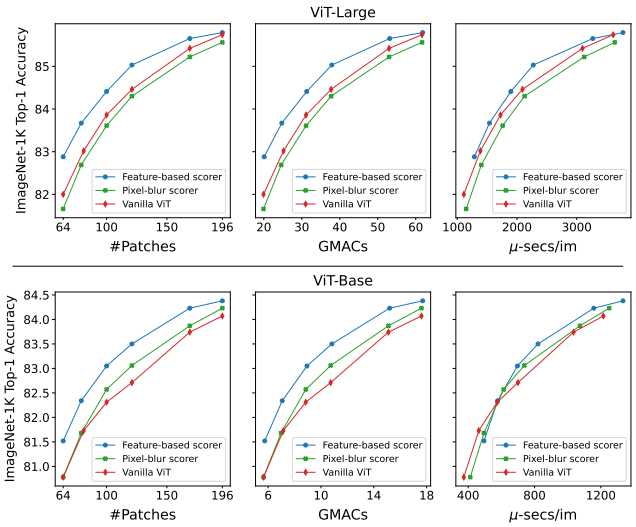

Figure 6. Accuracy vs compute for vanilla ViT and Quadformer models with different saliency scorers. Every point represents a model fine-tuned with a specific number of patches. We expect the performance of Quadformer and vanilla models to converge as #Patches approaches full resolution (256 patches). Throughput is measured on a single GeForce RTX 3090 GPU in mixed precision.

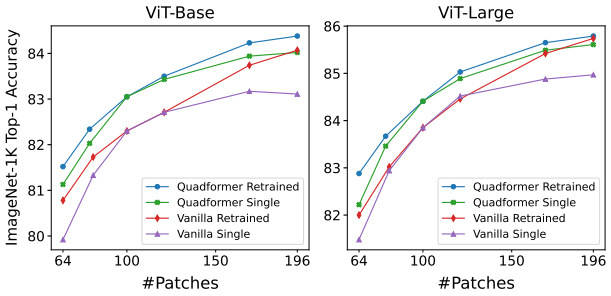

Figure 7. Inference-time compute-accuracy tradeoff for Quadformer models with a feature-based scorer and vanilla ViTs. "Retrained" lines show models that are retrained for each value of #Patches. "Single" lines show a single model (trained with 100 patches) evaluated with different #Patches. Quadformers are less sensitive to out-of-distribution input lengths, providing a better inference-time compute-accuracy tradeoff with a single model.

4.3. Main results

Using a feature-based scorer (§3.2), our Quadformer models consistently beat the accuracy of vanilla Vision Transformers by up to 0.79 (for ViT-Base) or 0.88 (for ViT-Large) absolute percentage points when controlling for the number of patches or GMACs, while using the exact same architecture (see Figure 6). Despite not using dedicated tools for accelerated inference, we also show gains when controlling for inference speed, beating vanilla ViT models for almost all values of #Patches by up to 0.42 (for ViT-Base) or 0.4 (for ViT-Large) absolute percentage points. The traditional pixel-based scorer used for image compression fairs much worse than our feature-based scorer, demonstrating the superiority of semantic meaning over surface details. Full results are provided in the supplementary material.

4.4. Inference-time compute-accuracy tradeoff

Both Quadformers and vanilla Vision Transformers can be trained with a certain number of patches and operate on inputs with a different number of patches, providing a way to control the compute-accuracy tradeoff of a single model during inference time. With Quadformers, we use a different number of Quadtree splits to produce tokenizations of different lengths, allowing high granularity as every split increases the number of patches by 3 – the split patch is replaced with its 4 children patches. With vanilla ViTs, we change the image size to a different multiple of the patch size, thus changing the total number of patches. When using vanilla ViTs with different image sizes, we scale the 2D patch positions to fit the range seen during training time, as we have seen better results when the inference-time position embeddings closely resemble those seen while training.

Figure 7 compares the inference-time compute-accuracy tradeoff of a single Quadformer model with a feature-based scorer and a single vanilla ViT model to versions of these models specifically trained for each number of patches. Quadformers are less sensitive to out-of-distribution input lengths, showing a lower accuracy drop with respect to their retrained counterparts, and providing a better inference-time compute-accuracy tradeoff with a single model.

4.5. Small Quadformers

Small Transformers pose an interesting challenge. On the one hand, weak models have the most to gain from high-quality saliency estimation, since they lack the capacity required to compensate for low-resolution images or mediocre patch selection. Quadformer-Small beats the accuracy of vanilla ViT-Small by up to 1.98 absolute percentage points when controlling for the number of patches, and by up to 1.54 points when controlling for GMACs. On the other hand, small Transformers are so fast that the runtime of the feature-based scorer is too costly compared to the total runtime (Figure 10) making it inefficient in terms of runtime-accuracy tradeoff, even compared to the weak, yet speedy, pixel-blur scorer (Figure 8).

Future work may find faster high-quality saliency estimators that would enable small Vision Transformers to use mixed-resolution tokenization efficiently. We note that many previous works dealing with efficient Vision Transformers [1, 16, 21, 36] do not report results for models that are as fast as ViT-Small, perhaps encountering similar issues with speeding up such fast models.

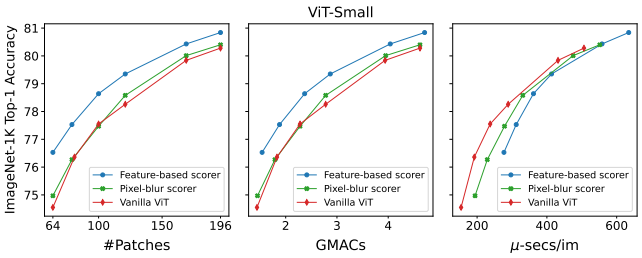

Figure 8. Accuracy vs compute for vanilla ViT-Small and Quadformer-Small models with different saliency scorers. Small Transformers pose an interesting challenge, being so fast that any tokenization overhead is significant.

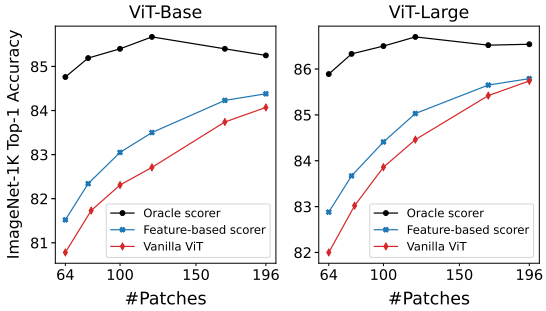

Figure 9. Quadformers with a Grad-CAM oracle scorer greatly surpass vanilla ViT models, suggesting there is considerable redundancy in standard ViT tokenization.

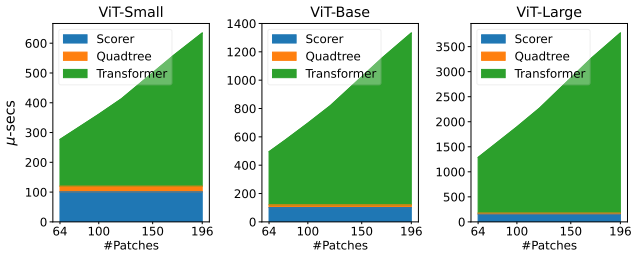

Figure 10. Quadformer forward pass runtime breakdown with a feature-based patch scorer. The fraction of time spent on tokenization changes drastically as the model and input length increase in size, from 44% for ViT-Small with 64 patches to 4% for ViT-Large with 196 patches.

Component		μ-secs	GMACs	$\frac{\mu-secs}{GMAC}$
Quadtree		19	0.0008	23,750
Patch Scorer	Pixel-Blur	19	0.0024	7917
	Feature-Based 256^2	157	0.166	945
	Feature-Based 192^2	101	0.094	1074
Transformer	ViT-Small 64 patches	154	1.44	107
	ViT-Base 121 patches	821	10.8	76
	ViT-Large 196 patches	3774	61.8	61

Table 1. Forward pass cost breakdown. Measuring actual runtime is especially important for comparison between Quadformers and vanilla ViTs since different components have very different runtime-to-GMACs ratios.

5. Analysis

5.1. Oracle Quadformers

To obtain a loose upper bound on the potential performance of ViTs with mixed-resolution tokenization, we train Quadformer models with a Grad-CAM oracle saliency scorer that has access to the true image label (§3.2). Our oracle models greatly surpass the performance of vanilla ViT models with the same number of patches – in some cases by about 4 absolute percentage points (Figure 9). Oracle Quadformers with 64 patches even beat vanilla ViTs with 196 patches despite using $\times 3$ less patches, suggesting there is considerable redundancy in standard ViT tokenization.

5.2. Runtime breakdown

The Transformer model and the saliency-based Quadtree tokenizer have very different runtime-to-GMACs ratios due to the different operations they use, with the tokenizer using a tiny number of GMACs compared to its runtime (Table 1). Therefore, we find that measuring actual runtime instead of settling for GMACs as the sole cost indicator is especially important when comparing our Quadformer models to vanilla ViT models.

The feature-based scorer requires 3 forward passes with a truncated ShuffleNetV2$\times 0.5$, composed mainly of group convolutions, depthwise convolutions, BatchNorms, and a channel-shuffle operation that has no GMAC cost but has a non-negligible time cost.

The Quadtree algorithm itself is very fast, though it has an especially high runtime-to-GMACs ratio, as it mostly requires indexing and reshaping operations that have no GMAC cost. Even though different numbers of patches require different numbers of splits, the bulk of the Quadtree runtime is spent preparing the input to the splitting phase and processing its output, making the Quadtree cost almost constant with respect to the number of patches.

The Transformer model is composed mainly of Attention layers, fully-connected layers and LayerNorms. Its runtime and GMAC cost depend heavily on the number of patches and the size of the model.

Notice that the fraction of time taken by patch scoring and Quadtree calculation becomes less and less significant as the model and input length increase in size (Figure 10), ranging from 44% for our lightest configuration to 4% for our heaviest.

Rank Correlation Coefficient	Similarity to Oracle: Feature-Based	Pixel-Blur	% Score$_{Feat}$ better
Kendall's τ	0.51	0.31	81%
Spearman	0.52	0.26	81%

Table 2. Average rank correlation between the patch rankings induced by the Grad-CAM oracle scorer and different realistic patch scorers, computed over the ImageNet-1K validation set. "% Score$_{Feat}$ better" measures how frequently the oracle ranking is closer to the feature-based scorer than to the pixel-blur scorer.

5.3. Patch scorer quality

The main way in which we assess the quality of different patch scorers is by measuring their effect on the downstream task, ImageNet-1K classification. Alternatively, we can measure scoring quality more directly by comparing the patch ranking induced by a Grad-CAM oracle scorer to the rankings induced by different realistic patch scorers (Figure 4). The oracle scorer is aware of the true image label, and uses the Grad-CAM algorithm which was built with the express purpose of saliency estimation, making it a good golden standard for patch ranking, as reflected in the high accuracy of oracle-based Quadformers (Figure 9).

For each image in the ImageNet-1K validation dataset, we calculate rank correlation coefficients between the oracle scores and the scores computed by the feature-based and pixel-blur patch scorers. We use rank correlations since the actual score values do not affect the Quadtree algorithm, only the relative ranking (see the $\arg\max$ operation in Algorithm 1). In Table 2 we report average rank correlation values and the fraction of images in the dataset for which the feature-based scorer was a better estimator of the oracle than the pixel-blur scorer, demonstrating the superiority of semantic representations over surface details.

5.4. Quadtree composition

Quadtree composition changes with the number of splits. As the iterative splitting process progresses, large patches are split into medium patches, which are in turn split into small patches. While the exact frequency of different patch sizes depends on the image content, we can get a sense of the resolution distribution by constructing Quadtrees over the entire ImageNet-1K validation set and measuring the average percentage of image area covered by each patch size (Figure 11). Note that the average resolution distribution depends on the ratio $\frac{\#Patches}{max(\#Patches)}$ and is almost invariant to the image size, which can help choose appropriate values for $\#Patches$ for different datasets, depending on the fraction of key information we expect the images to contain.

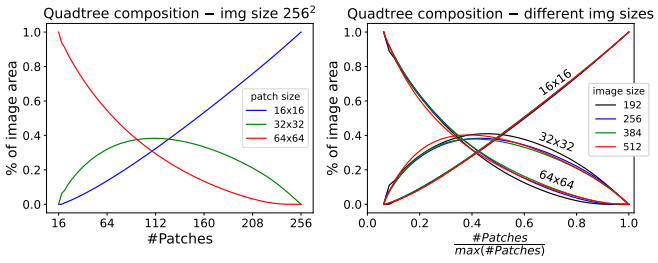

Figure 11. Quadtree composition changes with the number of splits. We measure the percentage of image area covered by each patch size to get a sense of the resolution distribution inside the image. The left plot shows the progression for our main image size. The right plot shows that this progression is almost invariant to the image size.

6. Conclusion

We have presented a novel tokenization scheme for Vision Transformers, replacing the standard uniform patch grid with a mixed-resolution sequence of tokens, where each token represents a patch of arbitrary size. We integrated the Quadtree algorithm with a novel feature-based saliency scorer to create mixed-resolution patch mosaics, making this work the first to use the Quadtree representations of RGB images as inputs for a neural network.

Through experiments in image classification, we have shown the capacity of standard Vision Transformer models to adapt to mixed-resolution tokenization via fine-tuning. Our Quadformer models achieve substantial accuracy gains compared to vanilla ViTs when controlling for the number of patches or GMACs. Although we do not use dedicated tools for accelerated inference, Quadformers also show gains when controlling for inference speed.

We believe that future work could successfully apply mixed-resolution ViTs to other computer vision tasks, especially those that involve large images with heterogeneous information densities, including tasks with dense outputs such as image generation and segmentation.

References

[1] Moab Arar, Ariel Shamir, and Amit H. Bermano. Learned queries for efficient local attention. *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 10831–10842, 2021. 2, 6

[2] Daniel Bolya, Cheng-Yang Fu, Xiaoliang Dai, Peizhao Zhang, Christoph Feichtenhofer, and Judy Hoffman. Token merging: Your vit but faster. *ArXiv*, abs/2210.09461, 2022. 1, 2

[3] Kashyap Chitta, José Manuel Álvarez, and Martial Hebert. Quadtree generating networks: Efficient hierarchical scene parsing with sparse convolutions. *2020 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 2009–2018, 2019. 2

[4] Mostafa Dehghani, Anurag Arnab, Lucas Beyer, Ashish Vaswani, and Yi Tay. The efficiency misnomer. *ArXiv*, abs/2110.12894, 2021. 5

[5] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *ArXiv*, abs/1810.04805, 2019. 2

[6] 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. *ArXiv*, abs/2010.11929, 2020. 1, 3, 5

[7] Patrick Esser, Robin Rombach, and Björn Ommer. Taming transformers for high-resolution image synthesis. *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 12868–12878, 2020. 2

[8] Raphael A. Finkel and Jon Louis Bentley. Quad trees a data structure for retrieval on composite keys. *Acta Informatica*, 4:1–9, 1974. 2

[9] Ross B. Girshick. Fast r-cnn. *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 1440–1448, 2015. 4

[10] Benjamin Graham, Alaaeldin El-Nouby, Hugo Touvron, Pierre Stock, Armand Joulin, Herv'e J'egou, and Matthijs Douze. Levit: a vision transformer in convnet's clothing for faster inference. *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 12239–12249, 2021. 1, 2

[11] Gregory M. Hunter and Kenneth Steiglitz. Operations on images using quad trees. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-1:145–153, 1979. 2

[12] Pradeep Kumar Jayaraman, Jianhan Mei, Jianfei Cai, and Jianmin Zheng. Quadtree convolutional neural networks. In *European Conference on Computer Vision*, 2018. 2

[13] Robert Jewsbury, Abhir Bhalerao, and Nasir M. Rajpoot. A quadtree image representation for computational pathology. *2021 IEEE/CVF International Conference on Computer Vision Workshops (ICCVW)*, pages 648–656, 2021. 2

[14] Lei Ke, Martin Danelljan, Xia Li, Yu-Wing Tai, Chi-Keung Tang, and Fisher Yu. Mask transfiner for high-quality instance segmentation. *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4402–4411, 2021. 3

[15] Taku Kudo. Subword regularization: Improving neural network translation models with multiple subword candidates. In *Annual Meeting of the Association for Computational Linguistics*, 2018. 1

[16] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows. *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 9992–10002, 2021. 2, 6

[17] Ningning Ma, Xiangyu Zhang, Haitao Zheng, and Jian Sun. Shufflenet v2: Practical guidelines for efficient cnn architecture design. In *European Conference on Computer Vision*, 2018. 5

[18] TorchVision maintainers and contributors. Torchvision: Pytorch's computer vision library. https://github.com/pytorch/vision, 2016. 5

[19] Tassos Markas and John H. Reif. Quad tree structures for image compression applications. *Inf. Process. Manag.*, 28:707–722, 1992. 1, 2, 4

[20] Donald Meagher. Octree encoding: A new technique for the representation, manipulation and display of arbitrary 3-d objects by computer. Technical Report IPL-TR-80-111, Rensselaer Polytechnic Institute, 1980. 2

[21] Lingchen Meng, Hengduo Li, Bor-Chun Chen, Shiyi Lan, Zuxuan Wu, Yu-Gang Jiang, and Ser Nam Lim. Adavit: Adaptive vision transformers for efficient image recognition. *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 12299–12308, 2021. 2, 6

[22] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019. 5

[23] Ilija Radosavovic, Raj Prateek Kosaraju, Ross B. Girshick, Kaiming He, and Piotr Dollár. Designing network design spaces. *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 10425–10433, 2020. 5

[24] Aditya Ramesh, Mikhail Pavlov, Gabriel Goh, Scott Gray, Chelsea Voss, Alec Radford, Mark Chen, and Ilya Sutskever. Zero-shot text-to-image generation. *ArXiv*, abs/2102.12092, 2021. 2

[25] Yongming Rao, Wenliang Zhao, Benlin Liu, Jiwen Lu, Jie Zhou, and Cho-Jui Hsieh. Dynamicvit: Efficient vision transformers with dynamic token sparsification. In *Neural Information Processing Systems*, 2021. 2

[26] Cédric Renggli, André Susano Pinto, Neil Houlsby, Basil Mustafa, Joan Puigcerver, and Carlos Riquelme. Learning to merge tokens in vision transformers. *ArXiv*, abs/2202.12015, 2022. 1, 2

[27] Gernot Riegler, Ali O. Ulusoy, and Andreas Geiger. Octnet: Learning deep 3d representations at high resolutions. *2017*

IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pages 6620–6629, 2016. 2

[28] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael S. Bernstein, Alexander C. Berg, and Li Fei-Fei. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115:211–252, 2014. 2, 5

[29] Michael S. Ryoo, A. J. Piergiovanni, Anurag Arnab, Mostafa Dehghani, and Anelia Angelova. Tokenlearner: Adaptive space-time tokenization for videos. In *Neural Information Processing Systems*, 2021. 2

[30] Ramprasaath R. Selvaraju, Abhishek Das, Ramakrishna Vedantam, Michael Cogswell, Devi Parikh, and Dhruv Batra. Grad-cam: Visual explanations from deep networks via gradient-based localization. *International Journal of Computer Vision*, 128:336–359, 2016. 5

[31] Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units. *ArXiv*, abs/1508.07909, 2015. 1

[32] Mannat Singh, Laura Gustafson, Aaron B. Adcock, Vinicius de Freitas Reis, Buğra Gedik, Raj Prateek Kosaraju, Dhruv Kumar Mahajan, Ross B. Girshick, Piotr Doll'ar, and Laurens van der Maaten. Revisiting weakly supervised pre-training of visual perception models. *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 794–804, 2022. 5

[33] Shitao Tang, Jiahui Zhang, Siyu Zhu, and Ping Tan. Quadtree attention for vision transformers. *ArXiv*, abs/2201.02767, 2022. 1, 3

[34] Maxim Tatarchenko, Alexey Dosovitskiy, and Thomas Brox. Octree generating networks: Efficient convolutional architectures for high-resolution 3d outputs. *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 2107–2115, 2017. 2

[35] Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Herv'e J'egou. Training data-efficient image transformers & distillation through attention. In *International Conference on Machine Learning*, 2020. 5

[36] Zhengzhong Tu, Hossein Talebi, Han Zhang, Feng Yang, Peyman Milanfar, Alan Conrad Bovik, and Yinxiao Li. Maxvit: Multi-axis vision transformer. In *European Conference on Computer Vision*, 2022. 2, 6

[37] Ashish Vaswani, Noam M. Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *ArXiv*, abs/1706.03762, 2017. 1

[38] Peng-Shuai Wang, Yang Liu, Yu-Xiao Guo, Chun-Yu Sun, and Xin Tong. O-cnn. *ACM Transactions on Graphics (TOG)*, 36:1 – 11, 2017. 2

[39] Wenhai Wang, Enze Xie, Xiang Li, Deng-Ping Fan, Kaitao Song, Ding Liang, Tong Lu, Ping Luo, and Ling Shao. Pyramid vision transformer: A versatile backbone for dense prediction without convolutions. *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 548–558, 2021. 1, 2

[40] Michael S. Warren and John K. Salmon. A parallel hashed oct-tree n-body algorithm. *Supercomputing '93. Proceedings*, pages 12–21, 1993. 5

[41] Ross Wightman. Pytorch image models. `https://github.com/rwightman/pytorch-image-models`, 2019. 5

[42] Tete Xiao, Mannat Singh, Eric Mintun, Trevor Darrell, Piotr Dollár, and Ross B. Girshick. Early convolutions help transformers see better. In *Neural Information Processing Systems*, 2021. 2

[43] Bo Xiong and Yuxin Wu. fvcore flop counting. `https://github.com/facebookresearch/fvcore/blob/2b14b2a025e60b0371a8509c42a2d45c821211c0/fvcore/nn/flop_count.py`, 2020. 5

[44] Jianwei Yang, Chunyuan Li, Pengchuan Zhang, Xiyang Dai, Bin Xiao, Lu Yuan, and Jianfeng Gao. Focal self-attention for local-global interactions in vision transformers. *ArXiv*, abs/2107.00641, 2021. 1

[45] Hongxu Yin, Arash Vahdat, José Manuel Álvarez, Arun Mallya, Jan Kautz, and Pavlo Molchanov. A-vit: Adaptive tokens for efficient vision transformer. *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 10799–10808, 2021. 2