A Broader impacts & limitations

Broader Impacts

NaViT enables training of vision transformers on variable size inputs, which has a profound impact on advancing adaptive computation research. By training models to handle various input size, we can explore adaptive computation techniques that dynamically adjust the computational resources based on the specific requirements of a given input. This flexibility opens up new avenues for implementing ideas that aim at adjusting allocation of compute and improving efficiency in vision tasks per input. Furthermore, NaViT computational efficiency unlocks the potential for scaling up pre-training of vision models. With the ability to handle different resolutions, models can effectively tackle more complex and diverse visual data, allowing for the development of larger and more powerful vision models.

Limitations

There is a wide range of applications that could benefit from a model capable of processing inputs of different resolutions, including OCR or document understanding with the use of vision models. Although we highlight the considerable advantages of employing NaViT and provide a comprehensive analysis of common computer vision tasks, we did not specifically investigate the benefits of NaViT in these particular applications. We consider this area as a priority for future research and follow-up work.

B Training details

B.1 Classification pretraining

The experiments and ablations in the paper are with ViT-B/32, ViT-B/16, and ViT-L/16. We use a reciprocal square-root learning rate schedule, with linear warmup and cooldown, with a maximum value of $8e-4$, and phases. We follow [18, 51] and use a higher weight decay of 3.0 on the head compared to the body's weight decay of 0.03 during upstream training to improve transfer to downstream tasks. For our experiments, we evaluated both NaViT and ViT models using configurations B/32, B/16, and L/16. Each ViT model was trained with varying compute budgets, with cooling down at different stages of training. We trained a corresponding NaViT model for each ViT size and computational budget, allowing us to perform "compute-matched" comparisons [25].

Table 2 presents the pretraining specifications for both ViT and NaViT models. During the pretraining phase, ViT models were trained using images of size 224×224. In contrast, NaViT models uniformly sampled a value, denoted as r, between 64 and 256 and resized the image to have a total of r^2 pixels while preserving the aspect ratio. Although training NaViT models on native resolutions is possible, we empirically discovered that sampling a resolution provides greater control over maximizing the number of examples observed during pretaining within a fixed computational budget while maintaining performance across different resolutions in downstream tasks. Additionally, by controlling the resolution, we can ensure efficient packing by tuning the sequence length and limit padding to less than 2%.

B.2 Contrastive pretraining

We use the 32000 token T5 [52] sentencepiece [53] tokenizer. By default, text sequences are truncated to a maximum length of 24. No token dropping is used for text. Models are trained under the same optimization regime as the classification models, but with a learning rate of 3×10^{-3}. Weight decay of 1×10^{-6} is applied consistently to all kernels in the model (no change for projection heads). By default, image side-lengths are sampled $\sim \mathcal{U}(64, R_{\max})$, and no other image augmentations are applied.

B.3 Packing algorithm

Packing of examples into sequences is done alongside batching. A simple greedy approach is used which adds examples to the first sequence with enough remaining space. Once no more examples can fit, sequences are filled with padding tokens, yielding the fixed sequence lengths needed for batched operations. Such simple packing algorithm can lead to a significant padding, depending on the distribution of length of inputs. There are several methods to address such limitations, like bin

Table 2: Pre-training details of ViT and NaViT with supervised classification.

Name	TPU Hours	Train Steps	Cooldown Steps	Sequence Length	Images Per Seq.	Batch Size	Training Images
ViT-B/32	1.4×10^{11}	1.0×10^{5}	1.0×10^{4}	49	1.0	$\approx4.0\times10^{3}$	4.0×10^{8}
	3.5×10^{11}	2.5×10^{5}	5.0×10^{4}	49	1.0	$\approx4.0\times10^{3}$	1.0×10^{9}
	7.1×10^{11}	5.0×10^{5}	1.0×10^{5}	49	1.0	$\approx4.0\times10^{3}$	2.0×10^{9}
	1.4×10^{12}	1.0×10^{6}	1.0×10^{5}	49	1.0	$\approx4.0\times10^{3}$	4.0×10^{9}
ViT-B/16	4.7×10^{11}	1.0×10^{5}	1.0×10^{4}	196	1.0	$\approx4.0\times10^{3}$	4.0×10^{8}
	1.1×10^{12}	2.5×10^{5}	5.0×10^{4}	196	1.0	$\approx4.0\times10^{3}$	1.0×10^{9}
	2.3×10^{12}	5.0×10^{5}	1.0×10^{5}	196	1.0	$\approx4.0\times10^{3}$	2.0×10^{9}
	4.7×10^{12}	1.0×10^{6}	1.0×10^{5}	196	1.0	$\approx4.0\times10^{3}$	4.0×10^{9}
ViT-L/16	9.8×10^{11}	1.0×10^{5}	1.0×10^{4}	196	1.0	$\approx4.0\times10^{3}$	4.0×10^{8}
	2.4×10^{12}	2.5×10^{5}	5.0×10^{4}	196	1.0	$\approx4.0\times10^{3}$	1.0×10^{9}
	4.9×10^{12}	5.0×10^{5}	1.0×10^{5}	196	1.0	$\approx4.0\times10^{3}$	2.0×10^{9}
	9.8×10^{12}	1.0×10^{6}	1.0×10^{5}	196	1.0	$\approx4.0\times10^{3}$	4.0×10^{9}
NaViT-B/32	1.4×10^{11}	9.8×10^{4}	1.0×10^{4}	64	5.41	$\approx2.2\times10^{4}$	2.1×10^{9}
	3.5×10^{11}	2.4×10^{5}	5.0×10^{4}	64	5.41	$\approx2.2\times10^{4}$	5.3×10^{9}
	7.1×10^{11}	4.8×10^{5}	1.0×10^{5}	64	5.41	$\approx2.2\times10^{4}$	1.0×10^{10}
	1.4×10^{12}	9.7×10^{5}	1.0×10^{5}	64	5.41	$\approx2.2\times10^{4}$	2.1×10^{10}
NaViT-B/16	4.7×10^{11}	9.3×10^{4}	1.0×10^{4}	256	4.87	$\approx1.9\times10^{4}$	1.8×10^{9}
	1.1×10^{12}	2.3×10^{5}	5.0×10^{4}	256	4.88	$\approx1.9\times10^{4}$	4.6×10^{9}
	2.3×10^{12}	4.6×10^{5}	1.0×10^{5}	256	4.88	$\approx1.9\times10^{4}$	9.2×10^{9}
	4.7×10^{12}	9.2×10^{5}	1.0×10^{5}	256	4.88	$\approx1.9\times10^{4}$	1.8×10^{10}
NaViT-L/16	9.8×10^{11}	9.7×10^{4}	1.0×10^{4}	256	4.88	$\approx1.9\times10^{4}$	1.9×10^{9}
	2.4×10^{12}	2.4×10^{5}	5.0×10^{4}	256	4.87	$\approx1.9\times10^{4}$	4.8×10^{9}
	4.9×10^{12}	4.8×10^{5}	1.0×10^{5}	256	4.87	$\approx1.9\times10^{4}$	9.6×10^{9}
	9.8×10^{12}	9.6×10^{5}	1.0×10^{5}	256	4.88	$\approx1.9\times10^{4}$	1.9×10^{10}

packing [7], which allows minimizing the padding. Here, in NaViT, since controlling the resolutions we sample, we can ensure efficient packing by tuning the sequence length and limit padding to less than 2%.

B.4 Sampling token dropping rates

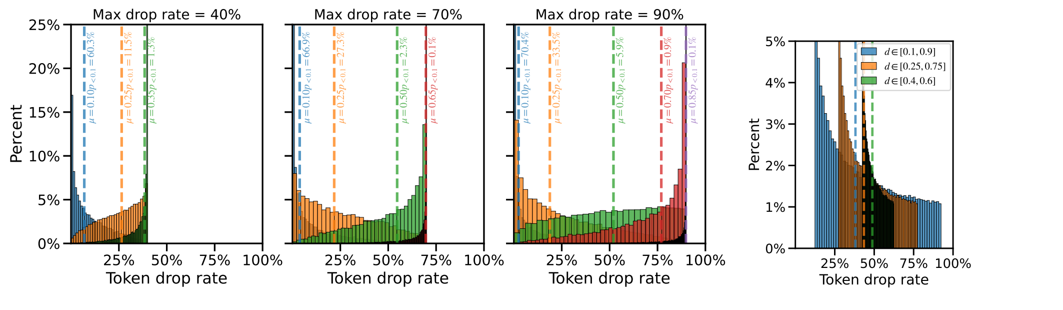
Sampling with a beta distribution We use a parameterisation based on the mean d_μ and standard deviation σ. We aim to sample dropout rate $d \in [0.0, d_{\mathtt{max}}]$, with some mean d_μ.

Accordingly, we sample $u \in [0,1] \sim \mathcal{B}(\alpha,\beta)$ and set drop rate $d = u \times d_{\mathtt{max}}$. α and β are set such that the mean of u is $u_\mu = \frac{d_\mu}{d_{\mathtt{max}}}$. The maximum supported variance for a beta distribution of mean u_μ is $u_\mu(1-u_\mu)$; we pick by default a variance $\sigma^2 = 0.3u_\mu(1-u_\mu)$, which we found to work well in practice. The resultant distributions of token dropouts for different settings of d_μ and $d_{\mathtt{max}}$ are shown in Figure 13a.

Sampling resolution-dependent dropping rates Given input data with sequence lengths ranging from $s_{\mathtt{min}}$ to $s_{\mathtt{max}}$, we sample dropout rate d from a truncated normal distribution $d \sim \mathcal{N}_{\mathtt{trunc}}(\mu, 0.02)$, where samples more than two standard deviations away from μ are rejected.

The mean of this distribution μ is set according to the minimum and maximum token dropping rates $d_{\mathtt{min}}$ and $d_{\mathtt{max}}$, and simply scales linearly with the sequence length s (such that $s = s_{\mathtt{min}}$ has $\mu = d_{\mathtt{min}}$ and $s = s_{\mathtt{max}}$ has $\mu = d_{\mathtt{max}}$.

Figure 13b shows example distributions of sampled drop rates given inputs with resolution $R \sim \mathcal{U}(64, 384)$, and different values of $d_{\mathtt{min}}$ and $d_{\mathtt{max}}$.


(a) Beta-sampled token drop rates parameterised by the mean μ and the max drop rate $d_{\texttt{max}}$ **(b)** Sampled resolution-dependent token drop rates

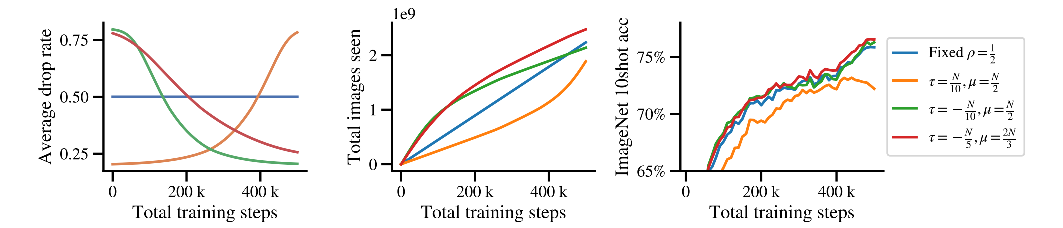


Figure 14: Decreasing the token dropping rate along training improves the ImageNet 10shot accuracy using the same pre-training resources. N is the total number of training examples seen with a fixed token dropping rate of $\rho = \frac{1}{2}$.

B.5 Scheduling token dropping rates

We experiment with a token dropping schedule which varies with total number of images seen. In particular, the rate applied for the n-th processed image during training is given by:

$$\rho(n;\rho_{\text{min}},\rho_{\text{max}},\mu,\tau) = \rho_{\text{min}} + (\rho_{\text{max}} - \rho_{\text{min}}) \cdot \sigma\left(\frac{n-\mu}{\tau}\right), \tag{1}$$

where σ represents the sigmoid function; ρ_{min}, ρ_{max} control the minimum and maximum dropping rate applied; and μ and τ control the shape of the schedule. We experimented with both increasing ($\tau > 0$) and decreasing ($\tau < 0$) schedules. In all cases we set $\rho_{\text{min}} = 0.2$ and $\rho_{\text{max}} = 0.8$. Figure 14 shows that, by decreasing the dropping rate throughout training, one can improve the final accuracy, at fixed training cost. Conversely, increasing the token dropping rate harms performance.

C Model information

C.1 Positional embeddings

Extending ViTs to variable input sizes necessitates rethinking positional embeddings added to every token after embedding. We considered several variants of positional embeddings, and evaluated them based on (1) the best performance model using them achieve within training distribution of input sizes; and based on (2) how well these models perform when evaluated on image sizes outside of the training distribution. Results and discussion of these experiments can be found in Section 3.4.

Broadly, we considered positional embeddings that varied along three axes: (1) whether they were learned, parametric or fixed; (2) whether they were absolute or fractional; and (3) whether they are factorized.

Absolute and fractional coordinates A natural way of indexing token within an image is to select *a priori* a maximum possible image side length (shared for width and height) maxLen, and to assign to token integer coordinates (x,y) based on their original location within the image. Embedding coordinates defined in this way allow models to consume images with resolutions up to $R = P \cdot \text{maxLen}$. However, when *learned* absolute coordinate embeddings are considered, extreme values of x and y must also be observed during training, which necessitates training on images with varied aspect ratios and limits models generalisation.

To alleviate the necessity of observing extreme aspect ratios and image size during learning of positional embeddings, we also consider fractional coordinates, which are normalized to the actual size of the input image and are obtained by dividing the absolute coordinates x and y above by the number number of columns and rows respectively, i.e. the corresponding side length. Doing this allows the model to observe extreme token coordinates during training, which intuitively should help with generalization to higher resolutions. However, this is accomplished at the cost of obfuscating the input images aspect ratio.

Factorized embeddings We further consider whether coordinates x and y should be embedded independently or jointly. In case of independent embedding, the two coordinates x and y are embedded independently, and their embeddings are combined via addition or by stacking. For joint embeddings and embedding for each position (x,y) is obtained directly.

Learned, parametric and fixed positional embeddings Finally, we also explored the relative benefits of fixed, learned and parametric embeddings. For fixed embeddings we followed [54] and used sinusoidal positional embeddings, and learned embeddings were implemented as in [1].

For parametric positional embeddings we followed [8] and used Fourier embeddings. Specifically, coordinates (x,y) were mapped using a single linear layer before applying $\sin$ and $\cos$ activations to them, and stacking the results to obtained the positional embeddings.

Experiments Because not all combinations of the above embedding choices are equally promising or natural, we experimented only with subset of them shown in Table 3 and Figure 10a.

Table 3: Classification of positional embedding experiments from Figure 10a.

Name	Coordinates	Type	Factorized
Learned 1D (ViT)	Absolute	Learned	No, position in flatted token sequence
Learned 2D (Pix2struct)	Absolute	Learned	No
Factorized abs. $(+)$	Absolute	Learned	Yes, sum
Factorized abs. (stack)	Absolute	Learned	Yes, stack
Factorized abs. $(\times)$	Absolute	Learned	Yes, product
NeRF abs.	Absolute	Parametric	No
Sinusoidal abs.	Absolute	Fixed	Yes, stack
Factorized frac. $(+)$	Fractional	Learned	Yes, sum
NeRF frac.	Fractional	Parametric	No
Sinusoidal frac.	Fractional	Fixed	Yes, stack

In sinusoidal and factorised embeddings experiments with fractional coordinates fractional coordinate embeddings were obtained from absolute coordinate embeddings via bilinear interpolation.

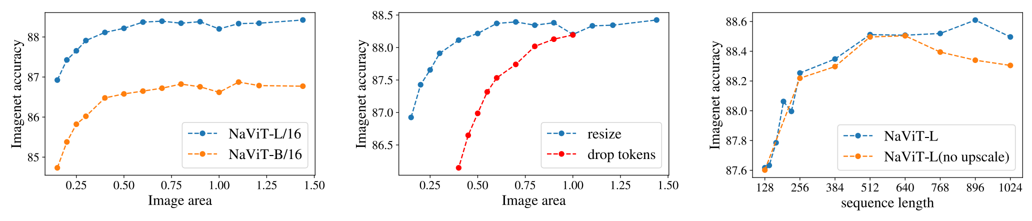
D Inference strategies

We performed various experiments to measure model quality for given runtime cost. The runtime can be tuned by changing the number of processed patches, or by using choosing different size of the model.

We firstly looked at how model quality changes in respect to decreasing area of the image compared to native resolution, presented in Figure 15a. We observed that on ImageNet [4] model retains most of the quality down to 40% of the image size. After that, the quality drastically decreases. On the other hand, increasing the size of the image have a diminishing return in quality. This can be directly compared with random token dropping as an alternative to resizing, which showed to be very ineffective way to decrease number of patches during inference - Figure 15b.

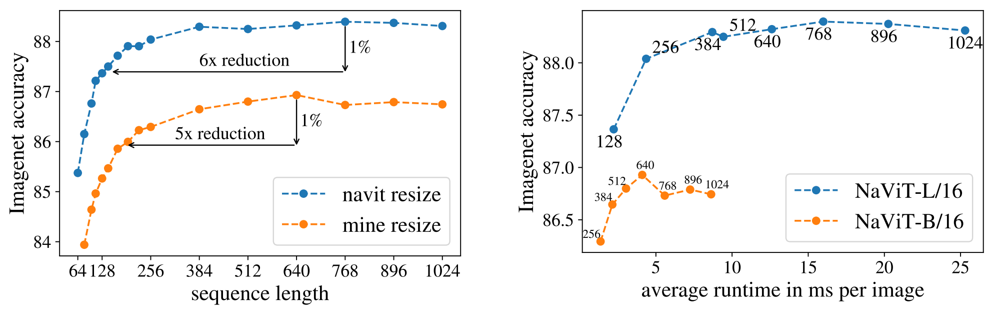
Please note that this highly depends on the native resolution of the images in the dataset - e.g. dataset with twice as big images than ImageNet can probably be safely resized to 20% of area.

A better way to quantify the performance is by giving a constant compute budget corresponding to number of patches. Figure 16a shows that resizing the image (while preserving aspect ratio) to 256 tokens retains most of the quality (within 0.3%). This corresponds to 256x256 area (given patch size of 16). At 128 tokens (181x181 area) the quality difference reaches 1% and drops significantly after that.



(a) **(b)** **(c)**

Figure 15: (a) The effect of resizing the image. (b) Dropping random tokens is ineffective way to decrease number of patches compared to resizing the image. Data from NaViT-L/16. (c) Given number of patches as compute budget, it is beneficial to upscale the image.



(a) **(b)**

Figure 16: (a) Quality on ImageNet in respect to number of patches (sequence length). (b) Runtime of models compared to the accuracy on ImageNet.

Here we also resized the image past its native resolution in case it already fit the given sequence length budget. We observed that it is beneficial to resize the image to the given sequence length past the native resolution to keep monotonic increase in quality, which is showed on Figure 15c.

Figure 16b presents the runtime of NaViT-L/16 and NaViT-B/16 for different sequence lengths. We can see that NaViT-L/16 at sequence length 128 is as fast as NaViT-B/16 with sequence length 512, while having almost 1% difference in quality.

E Cascades

Another strategy to be more compute efficient would be to assign more tokens (and thus FLOPs) to the examples deemed hard by the model. This is in particular interesting for bulk inference workloads, where one can amortize over large datasets and where only the total inference time matters.

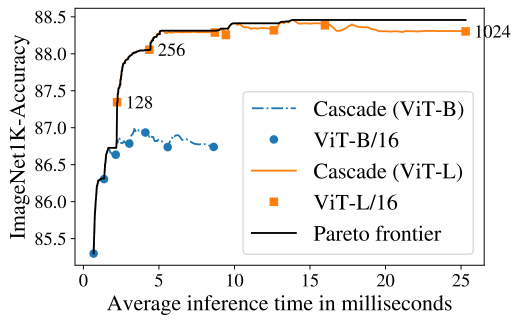


Figure 17: Performance of a model cascade versus the average inference time. The labels at the select points denote the number of tokens at that scale.

To evaluate the feasibility of this approach, we consider two sequence lengths n_1 and n_2 with respective inference times t_1 and t_2. Then, we (i) send all examples though the model with n_1 tokens, and send only the $\alpha \in (0,1)$-fraction deemed hardest (those with the smallest maximum probability) to the model with n_2 tokens. To have an input almost exactly fit into n_1 or n_2 tokens we perform and aspect ratio preserving resize. Hence, the total amortized inference time per-example is $t_i + \alpha t_2$, while the accuracy obtained by combining the accuracy of the first model on the $1-\alpha$ most-confident fraction of the data, and the performance of the more expensive model on the remaining data. By considering several pairs of models and varying α we obtain the plot in Figure 17. As we can see this strategy is indeed useful and provides not only the best performing models at given compute budgets, but because α is a real parameter one can obtain very fine-grained trade-offs.

F Calibration

To evaluate behaviour of the predicted uncertainties with scale, we compute the calibration error of a -B sized ImageNet-finetuned model (the -L model performs similarly). Note that these models were trained with sigmoid loss, i.e., the 1000 labels were predicted independently without enforcing that the probabilities should sum up to 1. As we varied the sequence of tokens per example between 128 and 1024, we obtained very stable calibration errors (top-1, using ℓ_1 and 30 buckets, i.e., the settings from [29]), which we present in Table 4.

Table 4: Expected calibration error on ImageNet-1K with varying sequence lengths.

Sequence Length	128	256	384	512	640	768	1024
Calibration Error	0.047	0.046	0.048	0.047	0.047	0.046	0.045

G Out of distribution evaluation

For ViT, we apply the "Crop" strategy from [17], namely an aspect-preserving crop of the central 75% of the image for ObjectNet and ImageNet-A, and square resize followed by a 87.5% central crop for the other datasets. We also apply a simple "Resize" strategy that does not crop the images. For NaViT, both the "Crop" and the "Resize" strategy do an aspect preserving resize of the target images.

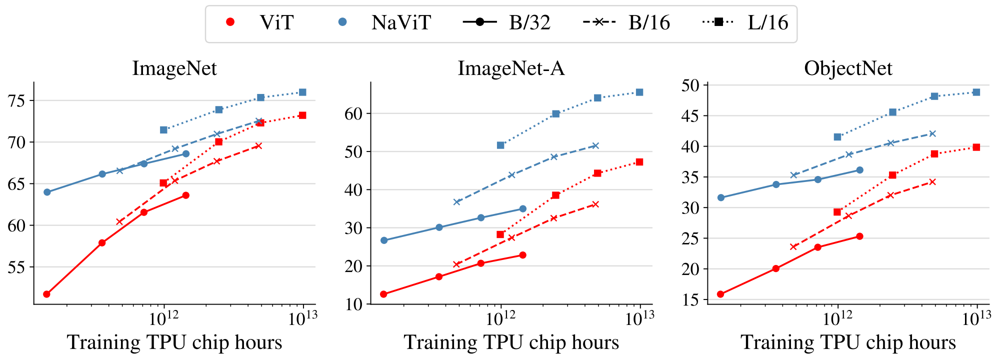


Figure 18: Same evaluation as in Figure 11, but without any special preprocessing of the images before the evaluation. Employing a simple resize (square for ViT, aspect preserving for NaViT) results in much better performance on datasets that have images with an extreme aspect ratio. Same data as in table Table 5.

H Fairness Signal Annotation

In Figure 19, we demonstrate that using native image resolution improves the performance of fairness signal annotation. Prior research has shown that metrics, such as group calibration, are vulnerable to labeling errors, particularly for underrepresented groups. Moreover, this problem persists even when accounting for label noise during training [30]. Thus, reducing the labeling error of fairness signals has

Table 5: Detailed results of out evaluation of pretrained models with a label-map (see Section 3.5). Same data as in Figure 11 and Figure 18.

			ImageNet		ImageNet-A		ObjectNet	
			ViT	NaViT	ViT	NaViT	ViT	NaViT
		Compute						
custom	B/32	1.4×10^{11}	54.4	63.5	14.7	32.7	32.9	41.2
		1.4×10^{12}	65.5	68.2	30.7	42.5	44.2	45.7
		3.6×10^{11}	60.2	65.6	22.0	37.0	38.0	43.5
		7.2×10^{11}	63.7	67.2	26.6	40.1	41.7	45.1
	B/16	1.2×10^{12}	66.4	68.5	37.3	52.3	47.2	48.4
		2.4×10^{12}	68.7	70.1	43.5	55.1	50.5	49.8
		4.8×10^{11}	61.7	66.0	27.0	44.2	41.3	45.9
		4.8×10^{12}	70.3	71.2	48.8	57.0	52.8	50.7
	L/16	2.5×10^{12}	70.7	73.6	51.5	65.5	53.3	55.0
		4.9×10^{12}	73.0	74.6	57.6	67.9	56.2	57.1
		9.9×10^{11}	66.4	71.1	39.2	58.6	47.6	52.1
		9.9×10^{12}	73.9	75.1	60.4	68.9	57.7	57.9
resize	B/32	1.4×10^{11}	51.7	64.0	12.6	26.7	15.9	31.6
		1.4×10^{12}	63.6	68.6	22.8	35.0	25.3	36.1
		3.6×10^{11}	57.9	66.2	17.2	30.1	20.0	33.8
		7.2×10^{11}	61.6	67.4	20.7	32.6	23.5	34.6
	B/16	1.2×10^{12}	65.4	69.2	27.4	43.9	28.7	38.7
		2.4×10^{12}	67.7	71.0	32.5	48.6	32.0	40.5
		4.8×10^{11}	60.4	66.5	20.4	36.8	23.6	35.3
		4.8×10^{12}	69.5	72.5	36.2	51.5	34.2	42.1
	L/16	2.5×10^{12}	70.0	73.9	38.5	59.9	35.3	45.6
		4.9×10^{12}	72.3	75.3	44.3	64.1	38.8	48.2
		9.9×10^{11}	65.1	71.5	28.2	51.6	29.3	41.5
		9.9×10^{12}	73.2	76.0	47.3	65.5	39.8	48.8

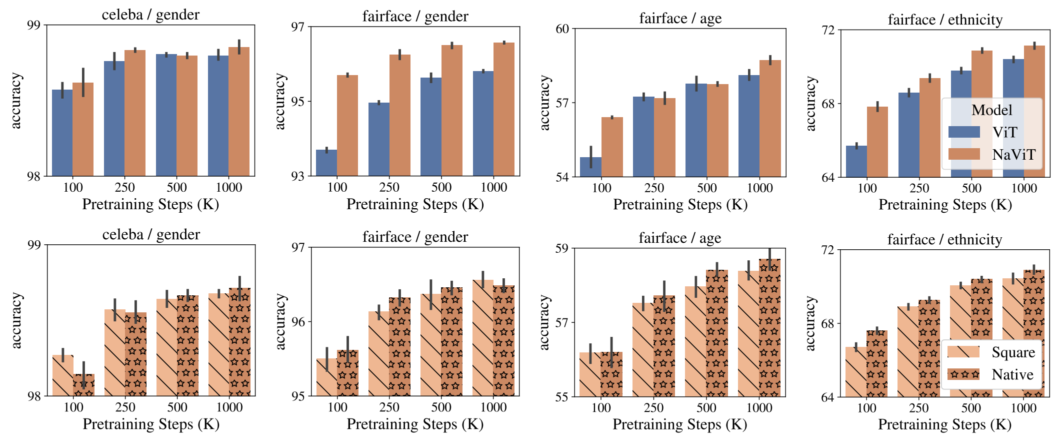


Figure 19: Summary of results of evaluating the accuracy of annotators trained on fairness-related signals using either NaViT-L/16 or ViT-L/16. TOP: NaViT offers better representations that improve the accuracy of annotators. BOTTOM: Using native aspect ratios in NaViT results in a higher performance when compared to resizing images to squares.

the potential of improving the reliability of bias mitigation and post-hoc auditing [31]. Nevertheless, we emphasize that while NaViT improves the annotation accuracy in these tasks, care must be taken in such situations since classifiers can be inaccurate and lead to a broad categorization of people that misidentifies real identities. We encourage readers to delve into the comprehensive work outlining such potential risks, e.g. [55, 56], for further insight. In assessing the technical capabilities of NaViT, our intent is not to promote or encourage their application in inappropriate contexts. Rather, our objective is only to illuminate these technical findings for scenarios where they may be considered beneficial, such as when measuring the level of diversity in a dataset or auditing/mitigating biases in predictive models. We strongly advocate for responsible AI use, maintaining that the benefits of technological advancements should not overshadow the importance of user safety and privacy. AI tools, including ours, should always be deployed judiciously, with a keen awareness of potential risks and a commitment to avoiding harm.

I Evaluation on model-vs-human OOD datasets on different resolutions

Just like NaViT, human visual perception works across flexible aspect ratios and resolutions (just imagine how strange the world would look like if we could only see it through a 224×224 pixel window!). We investigate how the ability to cope with variable resolutions affects performance on "model-vs-human", a benchmark of 17 challenging datasets [57].[1] For this purpose, we replicate the setup from Figure 6, but instead of evaluating ImageNet accuracy, we evaluate OOD accuracy on the model-vs-human benchmark.

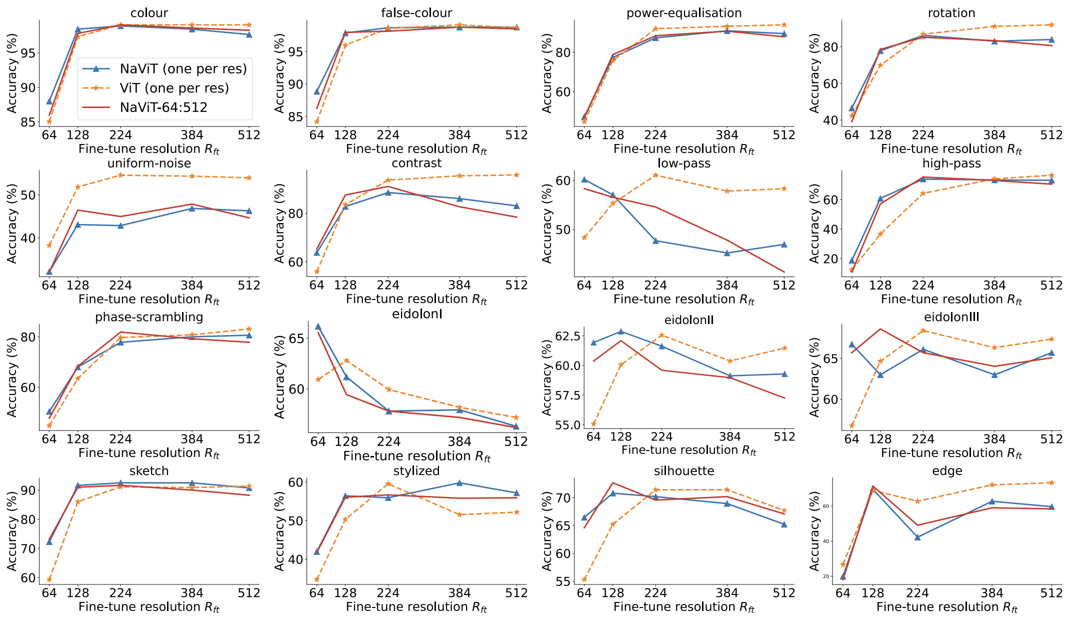


Figure 20: OOD accuracy on "model-vs-human" datasets across different fine-tuning resolutions. A single NaViT model trained on varying resolutions (red) performs roughly on par as fine-tuning one NaViT model per test resolution (blue). The ViT baseline (orange) is mostly worse than NaViT models for lower resolutions and mostly better for higher resolutions.

This corresponds to testing JFT B/16 models finetuned on ImageNet at various resolutions. The test dataset has a fixed 224×224 square resolution; thus we resize the test images to fit each model's fine-tuning resolution. Note that using square images has been standard practice designed for convolutional networks, but NaViT models no longer require square input, thus existing benchmarks are not tailored to those new possibilities. For datasets with multiple difficulty levels (such as different levels of blur), we average performance across levels while excluding levels that are too easy (not OOD) or too hard (human performance is close to chance), which follows the approach of [57] as explained in their "Appendix G Benchmark scores".

To ensure a fair comparison, we use models that are compute-matched for pretraining, and during fine-tuning, compute and data are identical for all models. The results of our comparison are shown in Figure 20. Overall, a single NaViT model trained on varying resolutions (red) performs roughly on par as fine-tuning one NaViT model per test resolution (blue).

The ViT baseline (orange) is mostly worse than NaViT models for lower resolutions and mostly better for higher resolutions. It may be worth noting that ViT models have a bit of an advantage in this comparison since they are fine-tuned on square images, whereas NaViT models are fine-tuned on flexible resolution images (preserving the image's aspect ratio) that have the same number of pixels, while not necessarily being square.

[1]For the purpose of our comparison, we exclude the "cue-conflict" dataset from the OOD evaluation, since there is no objective ground truth class in the case of images with a texture-shape cue conflict.