JEI Journal of Electronic Imaging

RESEARCH PAPER

MiDaS: a large-scale Minecraft dataset for non-natural image benchmarking

David Torpey,* Max Parkin, Jonah Alter, Richard Klein, and Steven James
University of the Witwatersrand, School of Computer Science and Applied Mathematics, Johannesburg, South Africa

ABSTRACT. Reinforcement learning (RL) has recently made several significant advances using video games as a testbed. While many of these games are relatively self-contained, there has been a recent push to develop agents capable of tackling massive, open-ended environments that are more reminiscent of the real world. One of the most popular of these platforms is Minecraft, but to attain human-level performance, agents must be able to learn, plan, and reason using high-dimensional image input. Commonly, an agent will attempt to extract lower-dimensional features that assist with downstream tasks. However, representation learning techniques have primarily been applied to real-world, natural image datasets, and it is unclear how these same methods might translate to an artificial world with non-natural images. We therefore present MiDaS, a novel large-scale Minecraft dataset featuring 36,000 labeled images across 60 classes. MiDaS contains information about both the blocks in the image, critical to solving the game, as well as auxiliary information such as time of day and biome. Further, we perform an evaluation of various models to benchmark performance on this new dataset. Since RL agents must be capable of learning features without labels, we include benchmarks of various self-supervised learning approaches on the dataset. Our results indicate that self-supervised methods perform best in the linear evaluation paradigm, particularly in low-label settings with a ResNet-based backbone, whereas ImageNet-pretraining assists more in the fine-tuning setting. The full dataset is available at https://github.com/MinecraftDataset/MiDaS.

 [DOI: 10.1117/1.JEI.33.1.013035]

Keywords: image dataset; self-supervised learning; benchmarks; representation learning

Paper 230944G received Aug. 8, 2023; revised Jan. 12, 2024; accepted Jan. 16, 2024; published Feb. 5, 2024.

1 Introduction

Minecraft is a 3D open-world sandbox, rich with complexity and possibilities for environmental manipulation. It is for this reason that interest in the domain has grown in recent years, particularly in deep reinforcement learning (RL).[1–3] By providing the necessary tools and resources, platforms like MineRL,[4] MineDojo,[5] and Microsoft's Project Malmo[6] have also facilitated further work in this field.

Despite the increased attention in this area, to the best of our knowledge, there exists no large-scale dataset of Minecraft images that features a wide selection of in-game objects. Such a dataset could prove useful when attempting to solve common in-game challenges like navigation and resource collection, or as a benchmark in various classification and detection frameworks.

*Address all correspondence to David Torpey, 674425@students.wits.ac.za

1017-9909/2024/$28.00

Therefore, we introduce MiDaS, a novel, large-scale Minecraft dataset featuring 36,000 labeled images across 60 classes, known as blocks. The curation process of MiDaS is designed to ensure good coverage of common in-game objects and blocks, as well as a suitable distribution across various dimensions, such as biome, time of day, and a block's natural position. The amalgam of these curation processes ensures that MiDaS serves as a useful benchmark dataset for Minecraft, and for artificial, non-natural image classification more generally.

Much progress has been made in recent years in machine learning with algorithms capable of efficiently learning very general, rich representations of data with a wide range of modalities, including images,[7–9] video,[10,11] and text.[8,12] In particular, images have an inherently extremely large dimensionality; even for relatively small images, the representational complexity quickly becomes prohibitive due the curse of dimensionality.[13]

Finding suitable low-dimensional representations of images that are useful across a wide range of tasks is still an open problem. Most modern approaches rely on some convolutional network-based architecture such as ResNet,[14] or some transformer-based architecture, such as the vision transformer (ViT).[15] Further, since labeled data is typically hard to scale due to the annotation bottleneck, much progress has been made in unsupervised visual representation learning algorithms—the most common paradigm being self-supervised learning (SSL). Techniques in this paradigm aim to learn useful, rich representations using only unlabelled data.[9,16] These pretrained networks can then be used for downstream tasks, including image classification and semantic segmentation.

To this end, we provide—through a systematic evaluation and benchmarking of models—guidance for what modern architectures and learning techniques work well for the inherently non-natural artificial domain presented in MiDaS. As part of this benchmarking, we perform various analyses based on the dimensions defined by the metadata obtained during curation, as well as by varying the number of labels available downstream, all in an effort to gain deeper insights into the performance of model variants in more granular and specific scenarios within the Minecraft world. Our contributions can be summarized as:

- MiDaS : a large-scale Minecraft dataset featuring 36,000 high-resolution labeled images across 60 classes, as well as numerous metadata fields [along with the data curation tool, Minecraft image processor (MIP)].
- A carefully designed curation process to ensure that the blocks within MiDaS contain good coverage of various dimensions, including time of day, biome, and natural position.
- A systematic benchmark of various models whereby we vary pretraining regime, architecture, and pretraining dataset. In addition, we include comprehensive analyses using some of MiDaS's metadata fields.

Our benchmarking results suggest that pretraining on the MiDaS dataset is the best approach if only fixed representations are required downstream. However, if the full backbone network can be fine-tuned in the downstream task, then pretraining on ImageNet is the preferable approach. Additionally, we find that self-supervised pretraining with a ResNet50 backbone is superior to supervised pretraining in either scenario and constitutes a generally reliable approach.

The rest of the paper is structured as follows. Section 2 presents previous work in the curation of similar datasets, as well as an overview of the current research landscape in visual representation learning. Section 3 then provides a comprehensive overview of the MiDaS dataset. Section 4 details the research methodology used during the evaluation and benchmarking of the models, with accompanying experiments and results presented in Sec. 5. Finally, we give some concluding remarks in Sec. 6.

2 Related Work

2.1 Image Datasets

The majority of common benchmark datasets used in computer vision research consist of natural images depicting real-world objects or scenes. These include ImageNet,[17] CIFAR100,[18] Caltech101,[19] COCO,[20] and OpenImages.[21] However, fewer benchmarks exist that contain images of non-natural or artificial scenes. Such benchmarks datasets are important for the

development of RL agents in game-like testbed environments, as well as for real-world applications, ranging from medical image analysis[22] to solar cell defect detection.[23]

Several existing Minecraft datasets exist. These include MineRL,[4] which contains millions of state-action pairs of human demonstrations and is used for training RL agents, as well as MineDojo.[5] While MineDojo contains a large amount of data pertaining to Minecraft, the data are typically unstructured, and there is no focus on learning useful representations of in-game objects. This is what prompted our curation and release of MiDaS, which enables the benchmarking of new models' representational power in terms of representing the plethora of objects and scenes within the common RL testbed environment of Minecraft.

2.2 Representation Learning

Most modern approaches to visual representation learning can be classed as either generative or discriminative. Generative approaches, such as generative adversarial networks[7] and variational autoencoders,[24] are typically more computationally demanding, as they usually operate directly in pixel space. Discriminative approaches have seen more interest from the research community of late due to their superior performance and more efficient training dynamics.

One of the most popular visual representation learning paradigms is known as SSL. Earlier approaches known as pretext task methods required a network to solve a manually defined proxy task, such as context prediction,[25] inpainting,[26] or solving image jigsaw puzzles.[27] In recent times, instance discrimination (ID) techniques have become the preferred approach to SSL.

Contrastive and non-contrastive approaches exist within ID, and they primarily differ in how they prevent representational collapse. Both techniques aim to pull the embeddings of positive samples together; however, contrastive approaches, such as SimCLR[9] and MoCo[28] prevent collapse using an InfoNCE-like[29] loss function to explicitly push away negative samples. Non-contrastive approaches, such as SwAV,[30] BYOL,[31] and DINO,[16] prevent collapse by regularizing the empirical covariance matrix of the embeddings.[32]

Since we focus on SimCLR and DINO in this work, we explain them here in more detail. The SimCLR architecture is shown in Fig. 1. An image x is sampled from the dataset, and two distinct views are generated by employing random data augmentations t and t', respectively, which are sampled from a (possibly infinite) set of random data augmentations. These two views are passed through an encoder network f, followed by a projection head (usually a multi-layer perceptron) g, to obtain the final latent embeddings z_i and z_j. The NT-Xent[9] loss is then computed and used to update the network weights. This loss encourages the latent embeddings from random views from the same image to be pushed together, and the embeddings of random views from different images to be pulled apart. Due to the implicit negative sampling that this loss function inherently relies on, the batch size must typically be large to produce a reliable estimate.

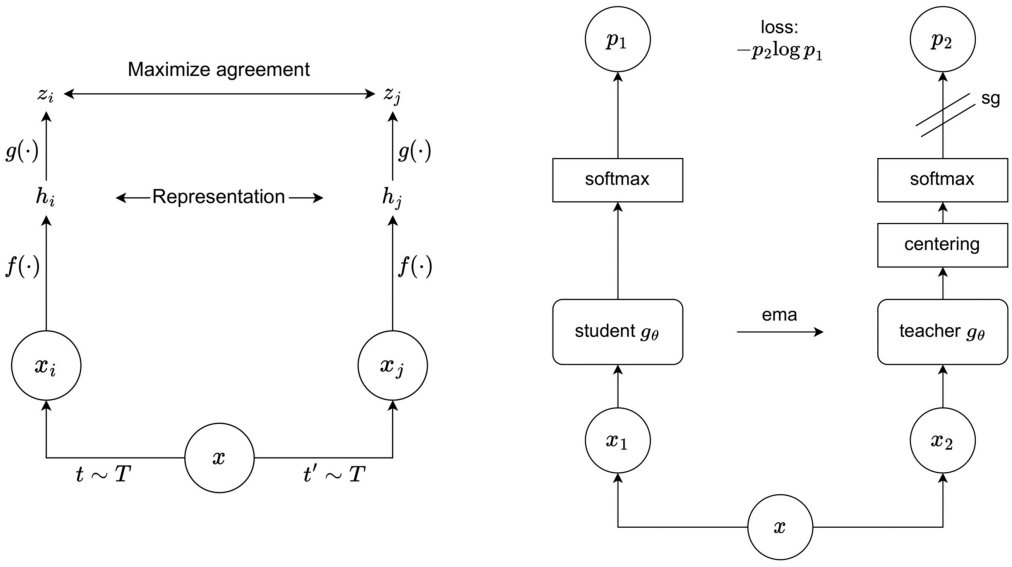

Fig. 1 Architecture diagrams for SimCLR (L) and DINO (R).

Figure 1 depicts the DINO architecture. Two random views are generated in the same way as in SimCLR. However, two separate networks (unlike the more common shared-weight approach in SimCLR)—the teacher and student—produce representations for their respective input views. The teacher embeddings are centered, and then both teacher and student embeddings are softmax-normalized. The cross-entropy between these softmax-normalized embeddings is then used as the loss. Importantly, only the student network is updated using backpropagation, whereas a stop-gradient is applied to the teacher network, and its weights are updated using an exponential moving average of the student network's weights. DINO is a form of self-distillation, a paradigm that has been widely applied in the visual SSL space.[31,33] It should be noted that for both SimCLR and DINO (and most SSL techniques), it is the encoder (and its associated representations) that are used for downstream tasks.

3 Dataset for Minecraft Objects

Minecraft features a large collection of discrete entities known as blocks (cube-like objects), as shown in Fig. 2. These are the fundamental units of the game and can be placed together to create complex structures. They are also combined in various ways to form new blocks. It is through their collection and manipulation that players interact with the Minecraft world. We now describe MiDaS, a novel Minecraft dataset consisting of 36,000 images with 60 block-level classes. A list of all blocks included in the dataset is included in the Appendix.

3.1 Data Collection

Before creating the dataset, we built a specialized annotation tool named the MIP (Fig. 3). MIP allowed us to automatically label and process a batch of images we had captured. In addition, it also allowed us to easily record metadata relevant to a particular set of images. This eliminated the need to manually process each image. The metadata and other important details are discussed further in Sec. 3.2.

The MIP software requires that two paths are specified: the source directory that stores the current batch of images, and the target directory where the images are transferred. The block input field (top-left in Fig. 3) determines the class folder that the images are transferred to in the target directory. If the class folder for the specified block does not exist, it is automatically

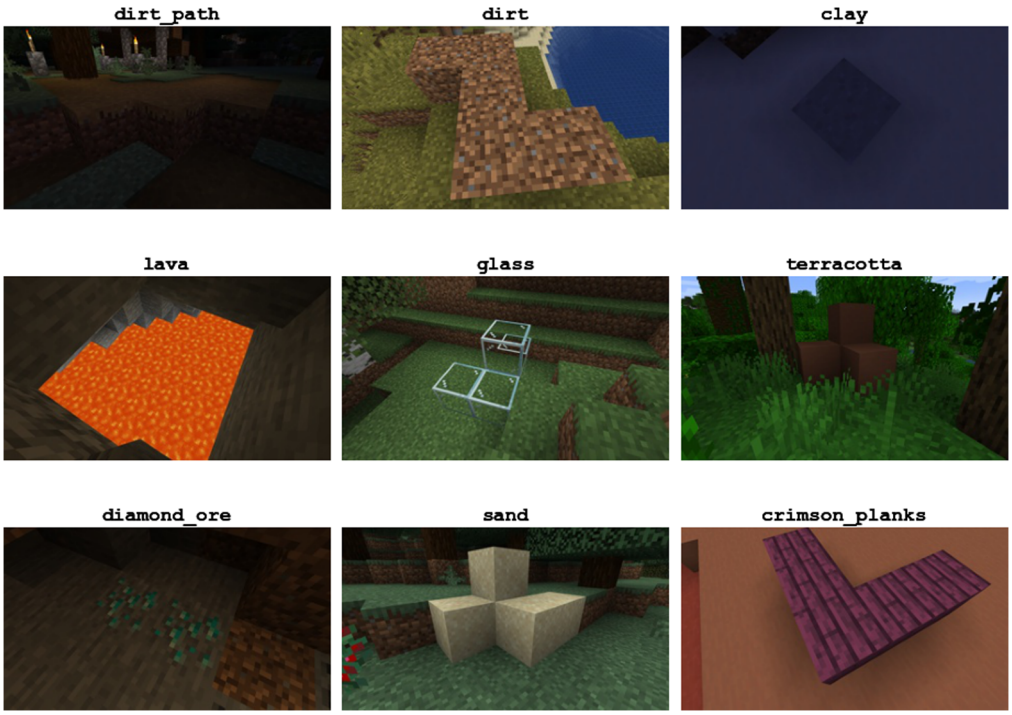

Fig. 2 MiDaS samples from nine different classes. Blocks are captured in different configurations, in various in-game biomes, at different angles and distances, and in differing lighting conditions.

Fig. 3 The MIP annotation tool.

created. The rest of the fields describe the metadata associated with a batch. As images are processed, a per-class CSV file records the specified input details. Any changes to a class folder via MIP are updated in this file. MIP also features delete functionality to remove processed images from the target directory, and a counting tool (bottom-right in Fig. 3) keeps track of the number of images in the source and target directories at any time.

When creating the dataset, images were obtained using Minecraft's in-game screenshot tool. This involved placing blocks in different configurations, or finding them in natural positions, capturing them from different perspectives, and then moving on to a new location. Without resizing the data, each image has spatial dimensions of 1920×1080 and a spatial resolution of 96dpi. The total dataset size is ~26 GB. All images were captured on version 1.17.1 on Minecraft: Java Edition.

The choice of which blocks to include was guided by the following criteria.

- Block rarity: we considered how likely it is for a player (or agent) to encounter the block in a standard playthrough, and prioritized the more common ones.
- Utilization in common training tasks: we included blocks that are frequently used when training an autonomous agent to perform certain tasks. For example, we include all types of logs and ores featured in the Treechop and ObtainDiamond challenges in MineRL, respectively.
- Natural generation: we aimed to include most blocks that spawn naturally in the game. These are blocks that a player will encounter in the Minecraft world without manually crafting and placing them.

There are hundreds of blocks in Minecraft and our choices are by no means exhaustive. Additionally, there are many important non-block items and resources in the game that a player will interact with. Therefore, we do not claim that MiDaS is fully representative of the Minecraft domain. However, our dataset includes most of the common blocks, and is diverse enough for interesting applications in the field, such a benchmarking performance of algorithms at recognizing the most common blocks that an agent is likely to encounter. Moreover, we welcome its expansion in the future and provide the MIP annotation tool at https://github/MinecraftDataset/MiDaS for this purpose.

3.2 Dataset Details

The data are distributed uniformly across the classes, with 600 images per block. To add variety to the data, a block is captured with an equal split of images across 10 different in-game biomes. This results in a total of 60 images per biome for a given block. Of these 60 images, 40 are in bright lighting conditions (day) and the remaining 20 in dark lighting conditions (night). If a block spawns naturally in the world, these images are included in the biome where it was found. However, there is no set number of these natural images that are included. While this is the common formula followed for most blocks, there are some exceptions. For these exceptions, the biome distribution was chosen on a per-block basis in a manner consistent with

Minecraft's mechanics. For example, since ore blocks only naturally occur underground (with the exception of coal ore, iron ore and, rarely, emerald ore), the majority of the 600 images for these blocks were captured in caves.

As mentioned above, metadata associated with each image in a class is included in a per-block CSV file. The important fields are described below.

- filename: the name of the PNG image as stored on disk. This follows the convention: <block>_<index>.png, where <block> is the name of the block and <index> is the position of the image within its class.
- block_details: extra information about the block in the image. This could be its recognized in-game state or some other high-level description that may be useful in downstream tasks.
- block_natural_position: if true, the captured block was not placed by a player. We note here that the environment around the block may still have been modified by a player.
- biome: the environment type in which the block is captured.
- dimension: there are three dimensions in Minecraft: the Overworld, the Nether, and the End. Each have distinctive terrains and features. Currently, MiDaS primarily contains blocks captured in the Overworld.
- time_description: whether the image was taken during the day or night. This field is not applicable when the block is underground or in the Nether (where there is no day-night cycle).

3.3 Dataset Statistics

We present some important statistics around MiDaS, its metadata, and the dataset distribution thereto. From Table 1, we can see the majority of the blocks in the dataset are outside of their natural position. This is primarily due to the sampling strategy employed during the curation of MiDaS, and enables more comprehensive benchmarking of models in recognizing the different block types. Table 2 shows the distribution of the time_description metadata flag across the entirety of MiDaS, and we note that approximately half of the images in the dataset contain blocks captured during daytime. Further, as mentioned previously, the 24% n/a are attributed to those blocks captured in regions with no notion of a day-night cycle (e.g., those blocks that are underground). We argue that these two tables show that MiDaS provides a good balance of block instances across these two dimensions.

Table 3 shows how certain statistics are distributed across the 21 unique biomes within MiDaS. Namely, the majority of biomes contain multiple different block types, such as deserts and plains. However, there are biomes with only one block type, such as gravelly_mountains and river. This is because in these biomes, there is only one common block type that can be sampled for curation of the dataset. For similar reasons, it is these biomes that contain fewer samples, whereas most of the biomes contain approximately 800 to 3000 samples.

Table 1 Dataset-level distribution of the block_natural_position metadata flag.

block_natural_position	#	%
True	9020	~25%
False	26,980	~75%

Table 2 Dataset-level distribution of the time_description metadata flag.

time_description	#	%
Day	18376	51%
Night	9072	25%
n/a	8552	24%

Table 3 Biome-level statistics: number of unique blocks and number of images per biome.

Biome	Number of blocks	Number of images
badlands	32	1900
basalt_deltas	7	602
birch_forest	42	2908
cave	19	4807
crimson_forest	7	649
dark_forest	41	3000
desert	43	2472
forest	3	547
gravelly_mountains	1	60
jungle	40	2940
mountains	1	97
nether_wastes	7	1055
plains	45	2689
river	1	7
savanna	42	3060
snowy_taiga	1	9
snowy_tundra	41	2409
soul_sand_valley	7	599
swamp	40	2343
taiga	42	3007
warped_forest	7	840

At a block-level, the vast majority of block types are sampled from ten different biomes. The typical split for a block's natural position is not balanced; blocks are either primarily in natural positions, or primarily in unnatural ones. For more details around the block-level distribution of these statistics, see Table 10 in Appendix A.

4 Benchmarking Methodology

4.1 Model Variants

We experiment with eight different model variants, where we vary the pretraining algorithm and pretraining dataset for each, as listed in Table 4. We experiment with both supervised pretraining (on ImageNet), and self-supervised pretraining (on ImageNet and on our MiDaS dataset), in addition to two of the most common architectures in modern vision: ResNet and ViT. Supervised and self-supervised pretraining, particularly on ImageNet, are two of the most common pretraining paradigms in the literature and thus serve as valuable benchmarks in this study. Additionally, self-supervised pretraining has recently been shown to outperform supervised pretraining on various downstream tasks in certain scenarios.[34]

After pretraining, we benchmark each model variant's performance on the downstream task of object classification using the MiDaS dataset (with labels, unlike during pretraining). To better understand the model variants at a more granular level, we investigate their behavior when varying the number of labels in the downstream task. In particular, we analyze the cases of 1%, 5%, and 100% of the labels being available to the model during this phase.

Table 4 Model variants considered in this work.

Variant name	Pretraining method	Pretraining dataset	Backbone architecture
Sup-RN50-Rdm	None	None	ResNet50
Sup-RN50-IN	Supervised	ImageNet	ResNet50
SimCLR-RN50-IN	SimCLR	ImageNet	ResNet50
SimCLR-RN50-MiDaS	SimCLR	MiDaS	ResNet50
DINO-RN50-IN	DINO	ImageNet	ResNet50
DINO-RN50-MiDaS	DINO	MiDaS	ResNet50
DINO-ViT-IN	DINO	ImageNet	ViT/S16
DINO-ViT-MiDaS	DINO	MiDaS	ViT/S16

For the self-supervised models, we experiment with two of the most popular and performant models—SimCLR[9] and DINO.[16] We choose these two models such that we account for both contrastive (SimCLR) and non-contrastive (DINO) approaches in our analyses. Further, it should be noted that all images of our MiDaS dataset or resized to a resolution of 320×180 prior to modeling for computational reasons.

It should be noted that the models used in our benchmarking were selected because they are common in the object classification realm (such as ImageNet-pretrained models). Further, we include self-supervised models since it has been shown that to outperform supervised pretraining in certain scenarios.[34]

4.2 Pretraining Details

For SimCLR pretraining, we use the Adam[35] optimizer with an initial learning rate of 0.03 and employ cosine decay. For DINO pretraining, we use the AdamW optimizer (for both the ResNet50 and ViT backbones) with an initial learning rate of 6.25×10^{-5} (also with cosine decay). We train both of these models from scratch for 500 epochs with a batch size of 32. We apply the standard set of augmentations for the generation of random views, as defined in the SimCLR work.[9] For Sup-RN50-IN, we use the pretrained weights available in PyTorch. For the ImageNet-pretrained self-supervised models, we use the weights available in TorchHub from the official repositories. Finally, for the MiDaS-pretrained self-supervised models, we train on the MiDaS dataset (without labels).

4.3 Downstream Task Details

We consider two types of phases for the downstream task, both of which are common in the literature: linear evaluation and fine-tuning. In both cases, we train on the MiDaS dataset, which is a 60-way object classification task. As mentioned above, we vary the number of labels during this phase to better understand model behavior in label-sparse settings.

For linear evaluation, we follow the conventional setup from the literature.[9,16,31] Namely, we take the pretrained encoder from the pretraining phase, freeze its weights, compute the embeddings for the training images, and train a multinomial logistic regression model on the representations using the L-BFGS optimizer. No data augmentation is performed during the computation of the embeddings.

For fine-tuning, we take the pretrained encoder from the pretraining phase and add a linear classification layer on top, and then fine-tune the entire model. In all cases, we train for 100 epochs with a learning rate of 0.001 with the Adam optimizer, except for the ViT-based models, where we use a smaller learning rate for improved training stability ($1e^{-4}$ for the MiDaS-pretrained model, and $1e^{-5}$ for the ImageNet-pretrained model). As with linear evaluation, we do not apply any data augmentations during fine-tuning.

All reported performance metrics for both linear evaluation and fine-tuning are an average of nine independent trials to obtain accurate performance estimates and confidence intervals.

5 Experiments

We perform analyses at a dataset level, class level, as well as using the metadata as obtained during the curation process detailed in Sec. 3.2. Importantly, much of our analyses focus on the 1% and 5% label sparsity settings, since these enable important and valuable insights into more granular details of model variants' performance and behavior.

5.1 Classification Performance

Below we discuss the results of both the linear evaluation and fine-tuning paradigms in terms of overall classification performance for each model variant. In Figs. 4–7, we represent supervised pretraining in red, SimCLR pretraining in blue, and DINO pretraining in green. Dotted bars indicate ImageNet pretraining, while diagonal lines represent MiDaS pretraining.

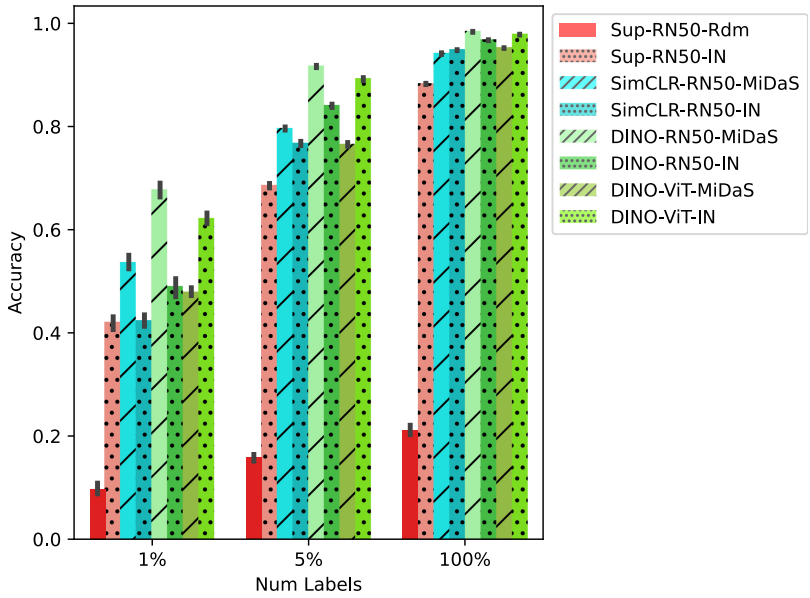

Fig. 4 Overall classification performance of the different model variants using the linear evaluation paradigm. Black bars represent the 95% confidence intervals.

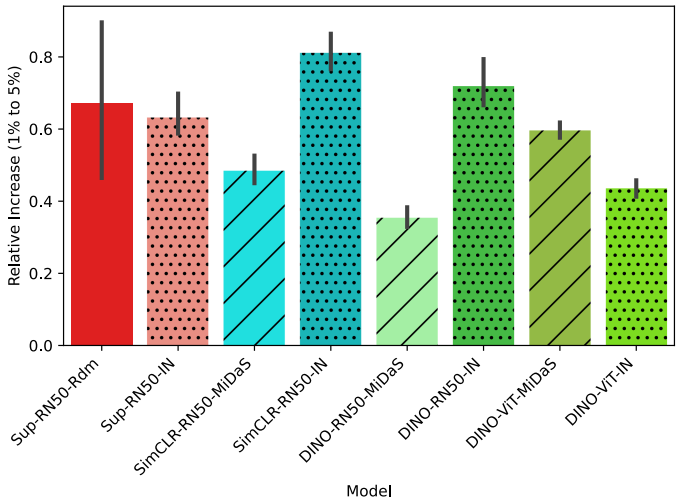

Fig. 5 Relative increase in performance when moving from 1% to 5% of labels using the linear evaluation paradigm. Black bars represent the 95% confidence intervals.

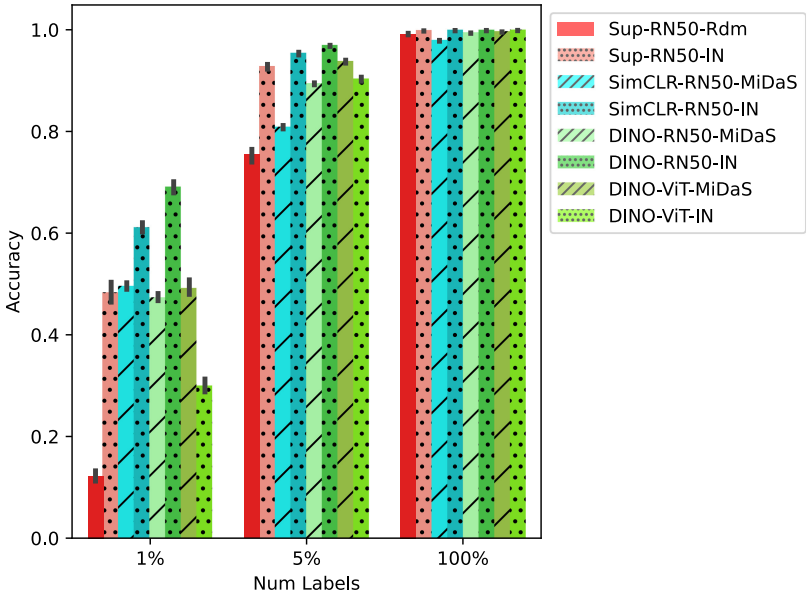

Fig. 6 Overall classification performance of the different model variants using fine-tuning. Black bars represent the 95% confidence intervals.

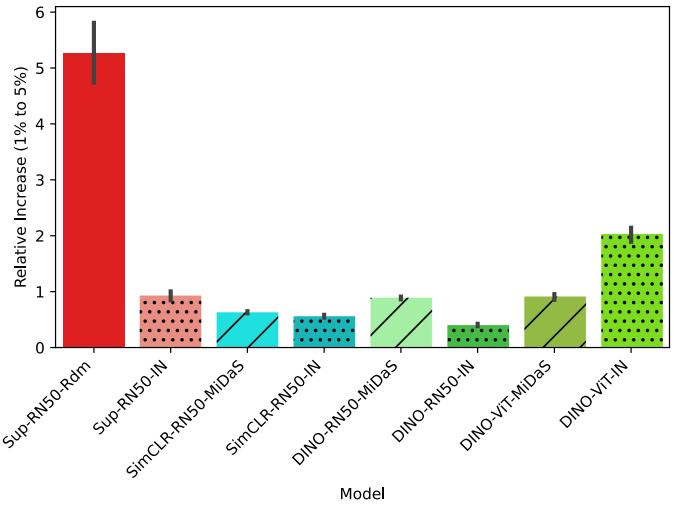

Fig. 7 Relative increase in performance when moving from 1% to 5% of labels using fine-tuning. Black bars represent the 95% confidence intervals.

5.1.1 *Linear evaluation*

The results in Fig. 4 demonstrate that performance differences are more easily recognizable in the low-label settings of 1% and 5% of labels. In general, the ResNet50-based models self-supervised on MiDaS perform best, with DINO (DINO-RN50-MiDaS) achieving the highest performance. Further, the ResNet50-based models self-supervised on ImageNet (SimCLR-RN50-IN and DINO-RN50-IN) perform worse than their MiDaS analogues. However, when the ViT architecture is pretrained on ImageNet, it outperforms the MiDaS-pretrained ViT. This is notable because it suggests that the ViT is potentially using the data within ImageNet more efficiently, resulting in more general embeddings (it is indeed known that ViTs learn and encode remarkably different information to ResNets[36]).

Understandably, linear evaluation with the Sup-RN50-Rdm variant (i.e., random weights/no pretraining) results in roughly the same performance no matter the number of labels. We also observe a generalization—to our artificial, non-natural domain of MiDaS —of the known

result that self-supervised pretraining on real-world, natural scenes outperforms supervised pretraining.[34] This is particularly evident in the low-label settings, where the self-supervised variants SimCLR-RN50-IN and DINO-RN50-IN outperform the supervised model Sup-RN50-IN.

To better understand the effect of additional labels on linear evaluation performance for each model variant, we plot the relative increase in performance when moving from 1% to 5% of labels. This can be seen in Fig. 5. It is clear that SimCLR-RN50-IN benefits the most from the additional labels, whereas DINO-RN50-MiDaS benefits the least. This somewhat makes sense, since SimCLR-RN50-IN only has context of ImageNet prior to linear evaluation, whereas DINO-RN50-MiDaS has sufficient context of MiDaS through the pretraining stage.

Suggestion: when treating the learned representations as fixed, perform self-supervised pretraining on the (in-domain) MiDaS dataset. However, if using a transformer-based backbone, perform self-supervised pretraining on a large, diverse dataset like ImageNet.

5.1.2 *Fine-tuning*

Similarly to Sec. 5.1.1, we will focus on the low-label settings in our analyses, since at 100% of labels (particularly for this paradigm), all models essentially solve the problem, achieving near 100% accuracy. Interestingly, from Fig. 6, we can see that the findings are almost a mirror image of those found during linear evaluation. That is, the ImageNet-pretrained models Sup-RN50-IN, SimCLR-RN50-IN, DINO-RN50-IN, and DINO-ViT-IN all perform better than their MiDaS-pretrained analogues. In particular, DINO-RN50-IN performs best, with the SimCLR analogue close behind. Further, again opposite to the linear evaluation findings, the ViT pretrained on MiDaS actually outperforms the ImageNet-pretrained ViT.

These findings suggest that the additional degrees of freedom—enabled by allowing the adaptation of the backbone weights during fine-tuning—allow the network to take advantage of the full utility of the rich, general ImageNet weights (particularly with the ResNet50 models). Even though the MiDaS-pretrained networks are tailored to the downstream task by being in-domain, this seemingly does not outweigh the benefit of the much larger scale of ImageNet, and the subsequent features that can be learned by pretraining on it. Additionally, since in SimCLR-RN50-MiDaS and DINO-RN50-MiDaS are pretrained on the same dataset as the downstream task, there may not be enough additional information for the network to learn during fine-tuning. This limitation is inherently not present in linear evaluation due to the frozen backbone and indeed somewhat explains the improved performance of MiDaS-pretrained models in that setting.

We show the relative increase in performance for all models in Fig. 7. Both the supervised methods, as well as the ViT-based methods, seem to benefit greatly from the additional labels when going from 1% to 5% of labels. In particular, in variant Sup-RN50-Rdm, where no pretraining is performed, there is over a 500% increase in performance with the additional labels. This makes sense since all other models started off the downstream task at a more beneficial location in weight space due to the associated pretraining step.

From Table 5, we can see with 1% of the downstream labels, SimCLR-RN50-IN sees the most benefit when unfreezing the backbone network (i.e., fine-tuning). Unsurprisingly, the model with no pretraining (Sup-RN50-Rdm) also sees much benefit from the fine-tuning, particularly in the 5% case. Interestingly, the MiDaS-pretrained ResNet50 models extract little benefit from the fine-tuning, and in some cases fine-tuning even hindering performance. We posit this is due to the features learned from MiDaS pretraining not being general or rich enough to fully take advantage of fine-tuning when compared with the much larger ImageNet dataset and its inevitably richer and more general features.

Suggestion: when the backbone network is fine-tuned as part of the downstream task, perform self-supervised pretraining on a large, diverse dataset like ImageNet. However, if using a transformer-based backbone, perform self-supervised pretraining on the (in-domain) MiDaS dataset.

5.2 Class-Level Analysis

The three best-performing classes for each model and downstream phase for both the 1% and 5% settings are shown in Tables 6 and 7. Similarly, the worst-performing classes are shown in Tables 8 and 9. It is clear that the best and worst-performing classes are roughly consistent for

Table 5 Relative difference in performance of linear evaluation versus fine-tuning for both the 1% and 5% settings. The 95% confidence intervals are provided. The model with the largest relative increase in bold, and the model with the lowest is italicized.

	1%	5%
Sup-RN50-Rdm	0.2888 ± 0.2239	3.7943 ± 0.295
Sup-RN50-IN	0.152 ± 0.0966	0.3521 ± 0.0148
SimCLR-RN50-MiDaS	-0.0745 ± 0.0306	0.0156 ± 0.0081
SimCLR-RN50-IN	0.4412 ± 0.04	0.2419 ± 0.0097
DINO-RN50-MiDaS	-0.3 ± 0.0232	*-0.0251 ± 0.004*
DINO-RN50-IN	0.4122 ± 0.0666	0.1523 ± 0.0055
DINO-ViT-MiDaS	0.0258 ± 0.0313	0.2246 ± 0.0112
DINO-ViT-IN	*-0.516 ± 0.0282*	0.0116 ± 0.0079

Table 6 Best performing classes for each model for fine-tuning.

	1%		5%	
Sup-RN50-Rdm	lava	0.721	lapis_ore	0.988
	netherrack	0.5	redstone_ore	0.977
	redstone_ore	0.331	lava	0.976
Sup-RN50-IN	redstone_ore	0.984	gold_ore	1.0
	gold_ore	0.965	redstone_ore	1.0
	lapis_ore	0.937	obsidian	0.999
SimCLR-RN50-MiDaS	coal_ore	0.955	lapis_ore	1.0
	lapis_ore	0.949	coal_ore	0.998
	lava	0.925	obsidian	0.991
SimCLR-RN50-IN	lapis_ore	0.933	emerald_ore	0.999
	obsidian	0.926	obsidian	0.999
	redstone_ore	0.912	red_mushroom_block	0.997
DINO-RN50-MiDaS	obsidian	0.855	obsidian	1.0
	lava	0.846	coal_ore	0.999
	netherrack	0.843	emerald_ore	0.998
DINO-RN50-IN	redstone_ore	0.968	obsidian	1.0
	gold_ore	0.958	redstone_ore	1.0
	obsidian	0.949	crafting_table	0.999
DINO-ViT-MiDaS	lapis_ore	0.963	gold_ore	1.0
	gold_ore	0.951	obsidian	1.0
	redstone_ore	0.938	emerald_ore	0.999
DINO-ViT-IN	redstone_ore	0.655	redstone_ore	0.992
	lava	0.653	emerald_ore	0.985
	emerald_ore	0.595	magma_block	0.983

Table 7 Best performing classes for each model for linear evaluation.

	1%		5%	
Sup-RN50-Rdm	lava	0.743	lava	0.801
	netherrack	0.539	netherrack	0.585
	water	0.224	glowstone	0.408
Sup-RN50-IN	gold_ore	0.916	lapis_ore	0.967
	torch	0.894	coal_ore	0.965
	coal_ore	0.862	redstone_ore	0.953
SimCLR-RN50-MiDaS	coal_ore	0.971	lapis_ore	1.0
	lapis_ore	0.966	nether_gold_ore	0.994
	netherrack	0.958	coal_ore	0.993
SimCLR-RN50-IN	lava	0.801	magma_block	0.961
	netherrack	0.77	lava	0.952
	birch_log	0.761	glass	0.946
DINO-RN50-MiDaS	obsidian	0.998	gold_ore	1.0
	gold_ore	0.961	obsidian	1.0
	coal_ore	0.96	redstone_ore	1.0
DINO-RN50-IN	red_mushroom_block	0.835	magma_block	0.973
	lava	0.806	bricks	0.965
	netherrack	0.777	glowstone	0.964
DINO-ViT-MiDaS	emerald_ore	1.0	emerald_ore	1.0
	gold_ore	1.0	gold_ore	1.0
	lapis_ore	1.0	lapis_ore	1.0
DINO-ViT-IN	glass	0.951	magma_block	0.997
	tnt	0.881	crafting_table	0.993
	bookshelf	0.88	obsidian	0.988

both linear evaluation and fine-tuning. The best-performing classes are mostly ore-like (lapis_ore, gold_ore, redstone_ore, etc.), or fairly distinctive objects, such as lava and obsidian. Interestingly, the results are fairly consistent across both architecture type and pretraining regime.

The worst-performing classes are mostly plank-like and wood-like objects (oak_fence, spruce_planks, jungle_planks, etc.), or objects that could reasonably be confused for something wood-like, such as bookshelf, chest or dirt_path. There is a high degree of visual similarity between these classes, and thus the models exhibit high entropy predictions for these classes and most often confuse them (particularly in low-label settings). In these settings, there is simply an insufficient amount of samples per class for the models to adequately learn features that discriminate between them, especially with linear evaluation. Notably, it is clear from both Tables 8 and 9 that with even only moderately more labels, performance significantly improves across all models.

Some of the best and worst-performing block types are shown in Fig. 8. It is clear from these examples that the plank-like blocks have a high degree of visual similarity, whereas the ore-like objects (on the top row) contain more discriminative features, such as the distinct color differences of the various ore regions.

Table 8 Worst performing classes for each model for fine-tuning.

	1%		5%	
Sup-RN50-Rdm	bricks	0.0	furnace	0.298
	chest	0.0	diorite	0.438
	clay	0.0	spruce_planks	0.447
Sup-RN50-IN	oak_fence	0.0	bed	0.662
	spruce_planks	0.041	spruce_planks	0.768
	glowstone	0.082	furnace	0.798
SimCLR-RN50-MiDaS	oak_fence	0.0	spruce_planks	0.264
	spruce_planks	0.035	jungle_planks	0.47
	dirt_path	0.108	acacia_planks	0.53
SimCLR-RN50-IN	oak_fence	0.0	spruce_planks	0.751
	spruce_planks	0.124	jungle_planks	0.817
	dirt_path	0.189	oak_planks	0.868
DINO-RN50-MiDaS	oak_fence	0.0	spruce_planks	0.352
	magma_block	0.043	jungle_planks	0.578
	spruce_planks	0.052	birch_planks	0.639
DINO-RN50-IN	oak_fence	0.0	spruce_planks	0.834
	spruce_planks	0.102	jungle_planks	0.888
	magma_block	0.245	dark_oak_planks	0.891
DINO-ViT-MiDaS	oak_fence	0.0	bed	0.74
	clay	0.061	spruce_planks	0.794
	spruce_planks	0.062	sandstone	0.809
DINO-ViT-IN	oak_fence	0.004	spruce_planks	0.571
	clay	0.011	jungle_planks	0.715
	spruce_planks	0.018	dirt_path	0.742

5.3 Biome Analysis

Figure 9 shows the performance of each model variant by biome. It is clear that certain biomes contain blocks that are significantly more difficult to classify than others. For example, the mountains, gravelly_mountains, and nether_wastes biomes tend to have far superior performance across all the models compared to the snowy_tundra, badlands, and desert biomes. The reason the latter biomes have poor performance in general is two-fold. First, the blocks that constitute these biomes in the dataset are visually similar, such as the various plank-like blocks. Second, these biomes tend to contain many block types, which makes it even harder to discriminate between them when there is such little inter-block variation. In contrast, the model variants perform well on the former biomes primarily because the notably higher inter-block variation for the blocks within these biomes (e.g., between the nether_gold_ore and lava blocks within the nether_wastes biome).

Additionally, we find model performance trends that somewhat corroborate the findings from Figs. 4 and 6. Specifically, for the linear evaluation paradigm, DINO-RN50-MiDaS performs the best across the vast majority of the biomes for both the 1% and 5% settings. Moreover, in most of the cases where this model variant does not perform best, another MiDaS-pretrained variant performs best, particularly for the 1% setting (such as with the basalt_deltas, cave, and

Table 9 Worst performing classes for each model for linear evaluation.

	1%		5%	
Sup-RN50-Rdm	crafting_table	0.0	crafting_table	0.01
	jungle_log	0.0	chest	0.012
	magma_block	0.0	bookshelf	0.016
Sup-RN50-IN	spruce_planks	0.025	spruce_planks	0.161
	magma_block	0.05	dark_oak_planks	0.261
	clay	0.062	jungle_planks	0.295
SimCLR-RN50-MiDaS	spruce_planks	0.048	spruce_planks	0.294
	oak_fence	0.064	jungle_planks	0.447
	honey_block	0.166	crimson_planks	0.48
SimCLR-RN50-IN	magma_block	0.043	spruce_planks	0.341
	spruce_planks	0.049	jungle_planks	0.49
	oak_fence	0.119	crimson_planks	0.494
DINO-RN50-MiDaS	magma_block	0.046	spruce_planks	0.326
	spruce_planks	0.07	jungle_planks	0.594
	jungle_planks	0.28	birch_planks	0.69
DINO-RN50-IN	spruce_planks	0.031	spruce_planks	0.307
	magma_block	0.078	dirt_path	0.605
	dirt_path	0.098	dark_oak_planks	0.633
DINO-ViT-MiDaS	dirt	0.071	jungle_planks	0.38
	oak_fence	0.078	spruce_planks	0.403
	spruce_planks	0.095	dirt	0.435
DINO-ViT-IN	spruce_planks	0.036	spruce_planks	0.47
	clay	0.109	jungle_planks	0.659
	jungle_log	0.203	dirt_path	0.735

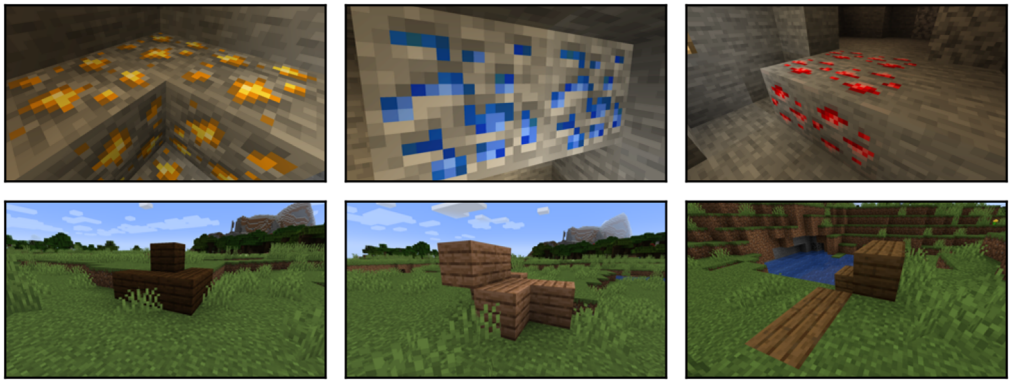

Fig. 8 Example images for some of the best-performing and worst-performing classes. Top row (best-performing) L-R: gold_ore, lapis_ore, and redstone_ore. Bottom row (worst-performing) L-R: dark_oak_planks, jungle_planks, and spruce_planks.

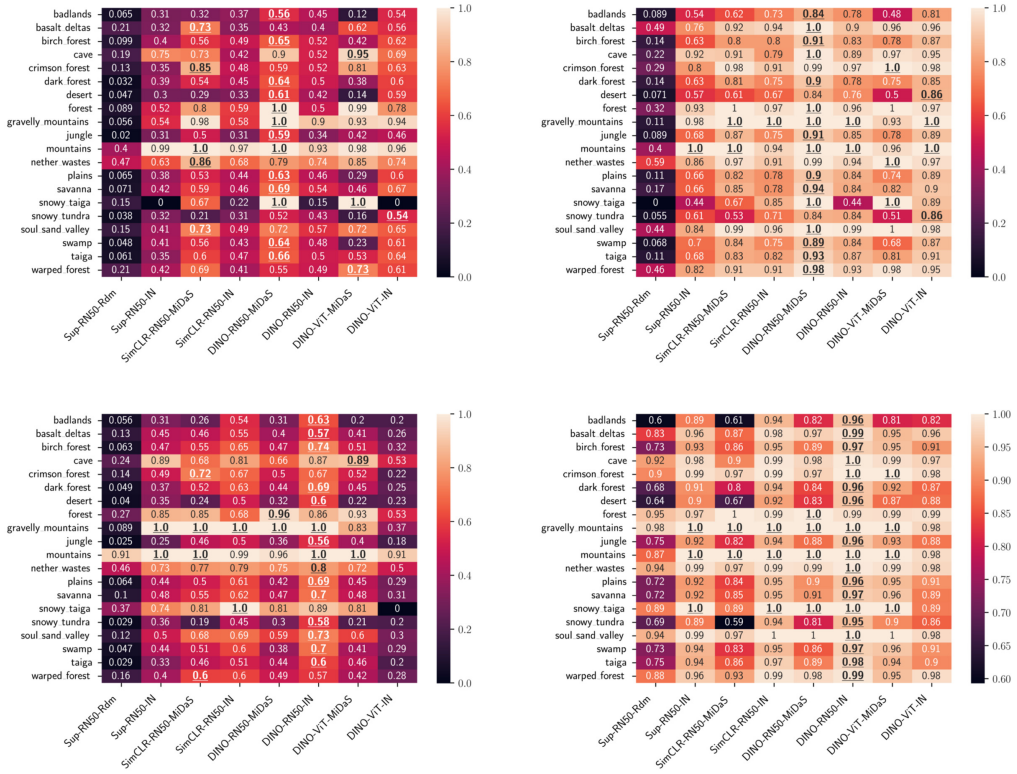

Fig. 9 Heatmaps depicting the performance of each model variant by biome (as defined in MiDaS 's metadata). The best-performing model for each biome is denoted by bold and underlined text. Top row, L-R: linear evaluation 1% labels and 5% labels; bottom row, L-R: fine-tuning 1% labels and 5% labels.

soul_sand_valley biomes). In the fine-tuning paradigm, the DINO-RN50-IN model performs best for the majority of the biomes for both label sparsity settings. Further, the MiDaS-pretrained models still perform relatively well in this fine-tuning setting. However, the benefit of pretraining on ImageNet is particularly clear in this setting, since the model can more easily take advantage of the general, rich features learned from that type of large-scale pretraining, even though the ImageNet dataset is out-of-domain with respect to the MiDaS downstream task. The benefit of MiDaS-pretraining is far more apparent when the encoder is frozen (i.e., fixed representations), as is the case in linear evaluation—the fact that the pretraining data is in-domain with respect to the downstream data seems to help significantly more in this setting.

Suggestion: if the downstream task requires the ability to differentiate between the high visually-similar objects, such as types of wood, more work is required (although the evidence suggests that ResNet50-based DINO pretraining generally performs well across the board and may serve as a useful direction in future work).

6 Conclusion

We introduced a large-scale, high-resolution Minecraft dataset (MiDaS) that can be used to benchmark new algorithms in preparation for a Minecraft RL testbed environment, and as a non-natural image object image classification benchmark in an artificial domain (along with the dataset, we release the curation tool used when creating MiDaS). MiDaS was carefully curated to ensure wide coverage of the pertinent and common objects within the Minecraft environment and includes sufficiently many samples to perform effective pretraining. Moreover, we perform a large-scale analysis of various models and pretraining methods to benchmark their performance on MiDaS. We analyze model behavior at a more granular level, including class level and using important metadata fields obtained during the curation of MiDaS. We note that in-domain pretraining on MiDaS results in the best performance for linear evaluation, whereas more general, large-scale pretraining on ImageNet performs best when fine-tuning downstream. The key

takeaway based on these results is to use a MiDaS-pretrained method if you require fixed representations downstream, and ImageNet-pretrained if you can utilize the full encoder network downstream. However, either way the key finding from the benchmark is that self-supervised pretraining with a ResNet50 backbone is a generally reliable approach from those evaluated.

7 Appendix A: Additional Dataset Statistics

We report additional statistics regarding other block-level flags obtained during curation of the dataset in Table 10. These include how often a block is found within its natural position within the images of the dataset, as well as the number of biomes each block type occurs in.

Table 10 Distribution of the block_natural_position flag for each of the 60 block types.

Block	% natural position	% unnatural position	Num biomes
acacia_log	0.93	0.07	1
acacia_planks	0.01	0.99	10
bed	0.05	0.95	10
birch_log	0.9	0.1	3
birch_planks	0.0	1.0	10
bookshelf	0.07	0.93	11
bricks	0.0	1.0	10
chest	0.03	0.97	10
clay	0.01	0.99	10
coal_ore	0.99	0.01	4
cobblestone	0.13	0.87	10
crafting_table	0.0	1.0	10
crimson_planks	0.0	1.0	9
dark_oak_log	0.9	0.1	1
dark_oak_planks	0.01	0.99	10
diamond_ore	0.04	0.96	1
diorite	0.1	0.9	10
dirt	0.0	1.0	10
dirt_path	0.13	0.87	10
emerald_ore	0.15	0.85	1
furnace	0.01	0.99	10
glass	0.0	1.0	10
glowstone	0.24	0.76	5
gold_ore	0.28	0.72	1
granite	0.1	0.9	10
grass_block	0.81	0.19	10
gravel	0.18	0.82	10
hay_block	0.14	0.86	10
honey_block	0.0	1.0	10

Table 10 (*Continued*).

Block	% natural position	% unnatural position	Num biomes
ice	0.1	0.9	10
iron_ore	0.68	0.32	1
jungle_log	0.9	0.1	1
jungle_planks	0.0	1.0	10
lapis_ore	0.19	0.81	1
lava	1.0	0.0	9
magma_block	0.17	0.83	10
melon	0.1	0.9	10
mossy_cobblestone	0.13	0.87	10
nether_bricks	0.04	0.96	5
nether_gold_ore	0.14	0.86	5
netherrack	1.0	0.0	10
oak_fence	0.08	0.92	11
oak_log	1.0	0.0	2
oak_planks	0.0	1.0	10
obsidian	0.04	0.96	10
pumpkin	0.03	0.97	10
red_mushroom_block	0.1	0.9	10
redstone_ore	0.05	0.95	1
sand	0.12	0.88	10
sandstone	0.1	0.9	10
slime_block	0.0	1.0	10
snow_block	0.04	0.96	10
spruce_log	0.96	0.04	1
spruce_planks	0.02	0.98	10
stone	0.24	0.76	10
terracotta	0.12	0.88	10
tnt	0.01	0.99	10
torch	0.14	0.86	10
warped_stem	0.33	0.67	5
water	1.0	0.0	10

Disclosures

There are no conflicts of interest for any of the authors of this manuscript.

Code and Data Availability

The data used in this research in made available at https://github.com/MinecraftDataset/MiDaS

Acknowledgments

We acknowledge the Mathematical Science Lab at the University of the Witwatersrand in aiding us to use the compute cluster for this research. Further, the authors acknowledge the Centre for High Performance Computing (CHPC), South Africa, for providing computational resources to this research project.

References

1. J. Oh et al., "Control of memory, active perception, and action in Minecraft," in *Int. Conf. Mach. Learn.*, PMLR, pp. 2790–2799 (2016).
2. C. Tessler et al., "A deep hierarchical approach to lifelong learning in Minecraft," in *Proc. AAAI Conf. Artif. Intell.*, Vol. 31 (2017).
3. S. Alaniz, "Deep reinforcement learning with model learning and Monte Carlo tree search in Minecraft," in *The 3rd Multidiscip. Conf. Reinforcement Learn. and Decis. Making* (2017).
4. W. H. Guss et al., "MineRL: a large-scale dataset of Minecraft demonstrations," in *Proc. 28th Int. Joint Conf. on Artif. Intell.*, pp. 2442–2448 (2019).
5. L. Fan et al., "MineDojo: building open-ended embodied agents with internet-scale knowledge," in *Adv. in Neural Inf. Process. Syst.*, Vol. 35, pp. 18343–18362 (2022).
6. M. Johnson et al., "The Malmo platform for artificial intelligence experimentation," in *Proc. 25th Int. Joint Conf. Artif. Intell.*, pp. 4246–4247 (2016).
7. I. Goodfellow et al., "Generative adversarial nets," in *Adv. in Neural Inf. Process. Syst.*, Vol. 27 (2014).
8. A. Radford et al., "Learning transferable visual models from natural language supervision," in *Int. Conf. Mach. Learn.*, PMLR, pp. 8748–8763 (2021).
9. T. Chen et al., "A simple framework for contrastive learning of visual representations," in *Int. Conf. Mach. Learn.*, PMLR, pp. 1597–1607 (2020).
10. C. Feichtenhofer et al., "SlowFast networks for video recognition," in *Proc. Int. Conf. Comput. Vis.*, pp. 6202–6211 (2019).
11. J. Carreira and A. Zisserman, "Quo vadis, action recognition? A new model and the Kinetics dataset," in *IEEE Conf. Comput. Vis. and Pattern Recognit.*, pp. 4724–4733 (2017).
12. T. Brown et al., "Language models are few-shot learners," in *Adv. in Neural Inf. Process. Syst.*, Vol. 33, pp. 1877–1901 (2020).
13. R. Bellman, *Dynamic Programming*, Princeton University Press, Princeton, New Jersey, USA (1957).
14. K. He et al., "Deep residual learning for image recognition," in *IEEE Conf. Comput. Vis. and Pattern Recognit.*, pp. 770–778 (2016).
15. A. Kolesnikov et al., "An image is worth 16x16 words: transformers for image recognition at scale," in *Int. Conf. Learn. Represent.* (2021).
16. M. Caron et al., "Emerging properties in self-supervised vision transformers," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV)*, pp. 9650–9660 (2021).
17. J. Deng et al., "ImageNet: a large-scale hierarchical image database," in *IEEE Conf. Comput. Vis. and Pattern Recognit.*, pp. 248–255 (2009).
18. A. Krizhevsky, "Learning multiple layers of features from tiny images," https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf (2009)
19. L. Fei-Fei, R. Fergus, and P. Perona, "Learning generative visual models from few training examples: an incremental Bayesian approach tested on 101 object categories," in *Conf. Comput. Vis. and Pattern Recognit. Workshop*, pp. 178–178 (2004).
20. T.-Y. Lin et al., "Microsoft COCO: common objects in context," *Lect. Notes Comput. Sci.* **8693**, 740–755 (2014).
21. I. Krasin et al., "OpenImages: a public dataset for large-scale multi-label and multi-class image classification," https://github.com/openimages (2016).
22. G. Varoquaux and V. Cheplygina, "Machine learning for medical imaging: methodological failures and recommendations for the future," *NPJ Digit. Med.* **5**, 48 (2022).
23. L. Pratt, D. Govender, and R. Klein, "Defect detection and quantification in electroluminescence images of solar PV modules using U-net semantic segmentation," *Renew. Energy* **178**, 1211–1222 (2021).
24. D. P. Kingma and M. Welling, "Auto-encoding variational Bayes," in *Int. Conf. Learn. Represent.* (2014).
25. C. Doersch, A. Gupta, and A. A. Efros, "Unsupervised visual representation learning by context prediction," in *IEEE Int. Conf. Comput. Vis.*, pp. 1422–1430 (2015).
26. D. Pathak et al., "Context encoders: feature learning by inpainting," in *Proc. IEEE Conf. Comput. Vis. and Pattern Recognit.*, pp. 2536–2544 (2016).
27. M. Noroozi and P. Favaro, "Unsupervised learning of visual representations by solving jigsaw puzzles," *Lect. Notes Comput. Sci.* **9910**, 69–84 (2016).

28. K. He et al., "Momentum contrast for unsupervised visual representation learning," in *Proc. IEEE/CVF Conf. Comput. Vis. and Pattern Recognit*. (2020).
29. A. V. D. Oord, Y. Li, and O. Vinyals, "Representation learning with contrastive predictive coding," arXiv:1807.03748 (2018).
30. M. Caron et al., "Unsupervised learning of visual features by contrasting cluster assignments," in *Adv. in Neural Inf. Process. Syst.*, Vol. **33**, pp. 9912–9924 (2020).
31. J.-B. Grill et al., "Bootstrap your own latent-a new approach to self-supervised learning," in *Adv. in Neural Inf. Process. Syst.*, Vol. 33, pp. 21271–21284 (2020).
32. Q. Garrido et al., "RankMe: assessing the downstream performance of pretrained self-supervised representations by their rank," in *Int. Conf. Mach. Learn.*, PMLR, pp. 10929–10974 (2023).
33. X. Chen and K. He, "Exploring simple Siamese representation learning," in *Proc. IEEE/CVF Conf. Comput. Vis. and Pattern Recognit.*, pp. 15750–15758 (2021).
34. L. Ericsson, H. Gouk, and T. Hospedales, "How well do self-supervised models transfer?" in *IEEE/CVF Conf. Comput. Vis. and Pattern Recognit.*, pp. 5414–5423 (2021).
35. D. P. Kingma and J. Ba, "Adam: a method for stochastic optimization," in *Int. Conf. Learn. Represent.* (2015).
36. M. Raghu et al., "Do vision transformers see like convolutional neural networks?" in *Adv. in Neural Inf. Process. Syst.*, Vol. 34, pp. 12116–12128 (2021).

David Torpey is a PhD student at the University of the Witwatersrand. He received his BSc, BSc Hons, and MSc degrees from the same institution. His research focuses on deep learning and computer vision, with a particular focus on self-supervised learning.

Max Parkin earned his bachelor's degree in computer science from the University of the Witwatersrand. Motivated by a passionate interest in computer graphics and machine learning, he pursued postgraduate research with a focus on computer vision and dimensionality reduction at the same institution. His academic pursuits reflect a dedication to exploring the intersection of these disciplines, thereby contributing to the advancement of the field.

Jonah Alter graduated from the University of the Witwatersrand, procuring a bachelor of science in computer science and a postgraduate degree specializing in computer vision. His core research interests are at the intersection between computer science and innovative gaming technologies. His academic journey, continued intrigue, and subsequent work in the field reflect his commitment to translating theoretical advancements into tangible applications in enhancing image processing and contributing to the landscape of those revolutionizing interactive entertainment.

Richard Klein is an associate professor at the School of Computer Science and Applied Mathematics, University of the Witwatersrand, South Africa, where he also received his PhD in 2017. He is a principal investigator at the PRIME Lab and RAIL Lab. He is one of the original founders of the Deep Learning Indaba. His interests involve computer vision and data-efficient learning.

Steven James is a senior lecturer at the University of the Witwatersrand, South Africa. He received his PhD from the same institute in 2021, where he was also the first African recipient of a Google PhD fellowship in machine learning. He is a principal investigator at the RAIL Lab, Africa's largest academic machine learning research group. His interests revolve around reinforcement learning and planning.