Transformer-based Monocular Depth Estimation using Defocus Cues

Venkat Subramanian
Adviser: Professor Jia Deng

Abstract

This paper details the development of a monocular depth estimation (MDE) machine learning model using defocus cues from a focus stack. Defocus-cue and focus stack based monocular depth estimation, formalized as Depth from Defocus (DfD), utilizes extra signal from camera optics to assist in otherwise ambiguous metric depth estimation. However, DfD has not been recently explored in the context of modern trends in the computer vision community, such as transformer-based model architectures and synthetic data training. The pipeline implemented in this project applies these modern computer vision methods to the DfD task to yield state-of-the-art results. In particular, I introduce a novel, memory efficient model architecture for the DfD task and find that this model, when combined with a synthetic data training pipeline, beats prior methods on various evaluation datasets.

1. Introduction 1.5

Depth estimation is a crucial component of many modern augmented reality, robotics, and self-driving tasks. In the case of self-driving, understanding 3D geometry of the car's environment of which depth plays an important factor, is critical to to maintaining safety and coordinating driving control. **Find self-driving car stats**. One useful component of furthering depth estimation capabilities in models for real-world applications is monocular depth estimation, or depth estimation using a single camera. Here, monocular depth estimation methods remain widely used for mobile-native applications, as stated in **Find sources here**. **Find AR stats** However, many monocular depth estimation AI models suffer from scale ambiguity issues and do not generalize well to unseen data and real-world images. The goal of my project is to beat state-of-the-art monocular depth

estimation methods using depth from defocus, where a model learns to predict the depth of a scene by receiving a set of images with different focus distances of the same scene as input. This particular method of using depth from defocus images is also interesting because jumping spiders are known to use defocus cues to do monocular depth estimation in a manner very similar to the depth from defocus method (Nolte et al.) **fix cites**. Ultimately, depth from defocus offers a cost-effective alternative to modern depth estimation methods, and if proven effective when compared to state-of-the-art methods, it can serve as a fallback depth estimation solution in autonomous vehicles and robots. DfD can also potentially boost depth estimation performance in "inherently monocular" cases, such as augmented reality applications on mobile devices.

The approach of this paper is to use synthetic training data wherever real, high-quality training data is lacking. We also introduce *DfDNet*, a transformer-based DfD model architecture motivated by recent trends in computer vision. We deliberately look to mitigate the higher inference times that larger transformer models have when compared to convolutional neural networks (CNNs). We will then compare inference times and performance of DfDNet against other comparable models to show that DfDNet and the training paradigm introduced in this paper offer a competitive datapoint on the Pereto frontier of DfD models.

2. Background and Related Work 4.5

2.1. Monocular Depth Estimation

Monocular depth estimation is loosely defined as the task of predicting depth of a scene using one camera. MDE is technically split into several branches, but when referring to MDE in this paper, I will specifically reference dense MDE, which means the task for a model is to predict the depth of every pixel from the camera, instead of predicting depth for a subset of pixels, which would be sparse MDE. The term "metric," refers to depth predictions that assign a distance typically in meters, whereas scale-invariant depth prediction refers to depth prediction that has no unit associated with it. Scale-invariant depth predictions from a model only indicate that the model believes that certain

points on the scene are a certain multiple closer or farther away from the camera than the other points in the scene, but the absolute "scale" of the entire scene remains unknown.

Include diagram of MDE (z axis distance from camera to object) Unfortunately, dense metric MDE in its traditional formulation is fundamentally limited by scale ambiguity of scenes. The issue of scale ambiguity is shown in **Figure Blank**. In particular, when looking at an image of a scene, a model can never determine with absolute certainty whether an object is small and close to the camera, or large and far away from the camera. Scale-invariant MDE models like DepthAntyhingV2 (**Cite**) mitigate this by having the model predict scale-invariant depth instead of metric depth. For DepthAnythingV2 depth predictions to be useable in real-world scenarios that require metric depth, the user must know the mean depth of the scene a priori and multiply every scale-invariant depth prediction by that value. Obviously, knowing the precise mean depth of a scene is still hard, so scale-invariant metric models fail to solve the metric MDE problem. Still, the authors of DepthAnythingV2 release both scale-invariant and metric MDE pretrained checkpoints for DepthAnythingV2, where the metric checkpoint is **Look up where the metric checkpoint comes from**

There have also been several recent works (**Cite MoGe-2, DepthPro, and maybe 1 more**) in metric MDE. These works broadly attempt to bypass the inherent scale ambiguity issue of MDE by training large transformer models with "large-scale training" to have the model implicitly learn to understand the general scale of objects in the real-world. For example, when looking at a scene with a car, a large metric MDE model would understand that the height of cars are generally 1.5-2.5 meters, which allows the model to triangulate how far the car, along with the objects nearby it, are from the camera. DepthPro does this through a separate branch to predict the focal length of the camera taking the image, while MoGe-2 does **Find what MoGe does**. Interesting developments have recently been made in the context of MDE for videos through models like Video Depth Anything, which involve models getting a video in the form of several image frames as input and predicting metric depth for each frame. The intuition for why video models that take several frames as input at a time would perform better than a single-frame metric MDE model applied on each video frame seprately is that depth predictions over a video must have temporal consistency. For

instance, a video of a dog running from far away to close to the camera should have a smooth transition of the depth prediction of the dog from far to close, whereas a metric MDE model may slightly mispredict the scale of the scene such that the dog is labeled with a large depth in one frame, a close depth in the next frame, and a far depth again in the next frame, which is temporally inconsistent and characterizes an undesirable phenomena referred to as flickering. We will lean on this intuition of temporal consistency in video metric MDE models and apply it to DfD.

Some metric MDE methods integrate camera intrinsics into the depth estimation to resolve field-of-view (FOV) ambiguity. CAM-Convs proposes a variant of a CNN for single-view metric depth estimation where the convolutions are conditioned on the camera intrinsics. ZeroDepth embeds the camera intrinsics into a transformer model such that the features for an image are "aware" of the camera intrinsics after a cross-attention block is applied between the image features and the camera intrinsics embedding vectors. Metric3D scaled AiF images and the corresponding depth map to a "canonicalized" space such that the model learns to predict in this canonical space and "undos" the canonicalization at evaluation or inference time.

Though we do not know exactly what deep networks are learning when trained for MDE, it is reasonable to assume that single-view metric MDE models are implicitly learning a "familiar metric size" of objects in the world and calculating metric distance from the camera to that object based on the number of pixels, or pixel size, of that object in the input image. However, if the focal length varies in the training data, then the FOV also varies, so the same object may have different sizes, which makes the single-view metric MDE problem much harder. Thus, if an image of a fixed size object from the same distance away from the camera has the same pixel size on the image, then a model could stably deduce the depth up to scale to the object and the relative depth of the scene. It is worth noting that integrating camera intrinsics via canonicalization or directly embedding the focal length does not resolve scale ambiguity entirely, but rather knowing the camera intrinsics allows a model to know that a point in an image maps to a particular ray direction extending out of the pinhole camera. The exact point along the ray that corresponds to the distance of the point from the camera is still unknown and must be estimated via some other signal like familiarity with the

general size of objects in a scene or defocus cues. Cite: https://arxiv.org/html/2312.06594v2

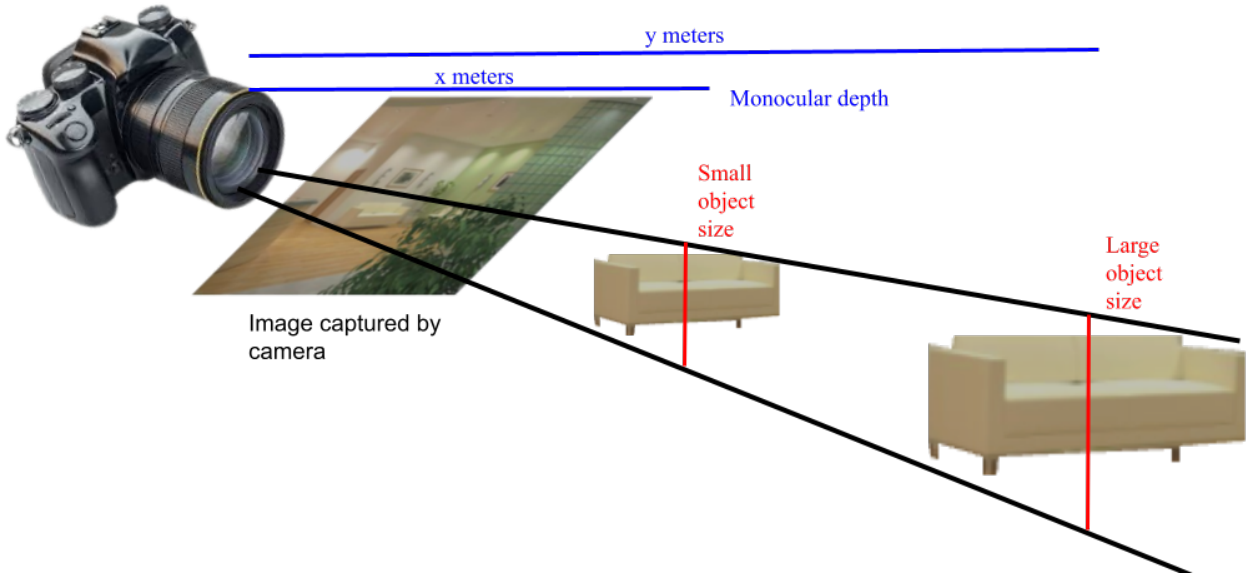

Figure 1: This is a gray image.

2.2. Depth from Defocus

Depth from defocus is a subfield of MDE that relies on defocus, or blur, cues and camera optics to ideally boost depth estimation performance in some settings. In particular, DfD models take in a focus stack, or set of images taken with all intrinsics being held constant and the only difference being the focus distances (FD) that the images were taken with. A key formula dictating the relationship between camera intrinsic parameters, object distance, focus distance, and defocus is the Circle of Confusion (CoC) formula, which is derived from the thin-lens approximation equation:

$$\text{CoC} = \frac{|d - d_f| \cdot f^2}{2 \cdot a \cdot d \cdot (d_f - f)} \tag{1}$$

Here, d is the depth of a particular point in the scene projected onto the camera plane, d_f is the FD of the image, f is the camera's intrinsic focal length, and a is the F-number. **This paper: https://onlinelibrary.wiley.c** goes onto describe how to approximate the circle of confusion formula using Gaussian convolutions to efficiently generate a focus stack for specified focus distances using a ground-truth depth map and all-in-focus image. **Figure 2** shows an example focus stack with different focus distances,

where one can notice an object in a scene appears in focus in an image in the focus stack where the FD corresponding to the image matches the ground-truth depth of the object. **Figure 2** also shows that focus stacks where the only difference is the F-number (a) has noticeably fewer defocus cues, meaning that an intuition for DfD is that the larger the aperture opening is (smaller a), the greater the defocus cues, and better DfD methods should generally perform over traditional MDE methods.

There is work done previously in the field of depth from defocus. Carvalho et al. shows that a single defocus image can potentially provide better depth cues and improved accuracy for computer vision models than an input being an all-in-focus image. Lu et al. and Si et al. both introduce depth from defocus models that use convolution-based model architectures. Notably, Si et al. released code that they used to generate synthetic focal stacks from all-in-focus images and ground-truth depth maps using the afformentioned Gaussian convolution approximation of the CoC formula, as shown in **Figure 3**

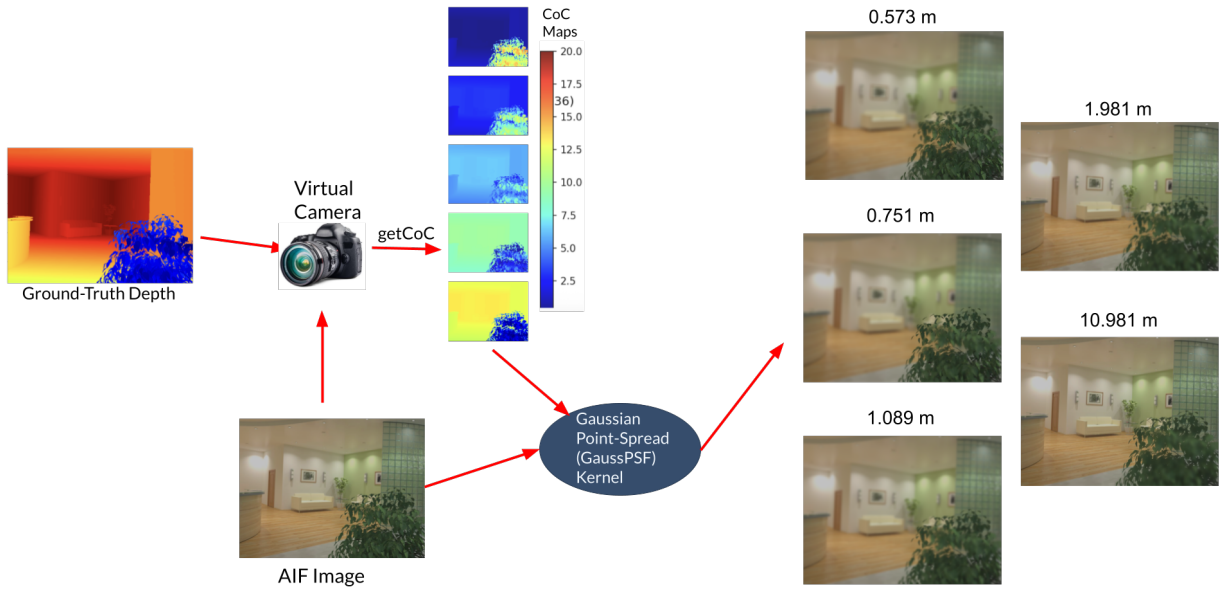

Figure 2: This is a gray image.

This synthetic focus stack generation method is particularly valuable for scaling the training data of DfD models. Traditional MDE models only require AiF images and corresponding ground-truth depth maps to train, and numerous datasets like Hypersim, ARKitScenes, TartanAir, VKITTI, and others exist meeting this requirement (**Cite**). However, there are no large-scale training datasets for

DfD models since no large dataset exists with focus stacks, focus distances, and ground-truth depth maps. DDFF-12 (**Cite**) does release a training subset of 400 images of the DDFF-12 dataset meeting these requirements, but we find that the quality of this dataset is low for several reasons. First, there are many holes in the ground-truth depth map. The diversity in the scenes captured in this dataset is also lacking. Furthermore, 400 images is simply not enough to train a modern transformer model. For example, DepthAnythingV2 trains on tens of thousands of images **Check this and cite**, as do other comparable models (**Cite**). Table **BLANK** shows the available training datasets in the literature in detail.

2.3. Transformers

This project also looks to build upon previous depth from defocus models by introducing a transformer-based (not convolution-based) approach that builds upon the recent shift in computer vision towards vision transformers (Dosovitskiy et al.) and leveraging rich pre-trained backbones from other work (Oquab et al.).

Consider also the recent developments with the encoder-decoder transformer architecture. At a high-level, encoder-decoder models are split into the encoder, which takes an input, patches it into tokens, and transforms those tokens into vectors in a high-dimensional space where similar original tokens are closely aligned embedding vectors of the encoder. Then, the decoder operates on these embedding vectors to make the type of prediction required for the model. A classic encoder-decoder architecture for MDE is that of DepthAnything, which uses the popular DINO encoder (**Cite**) and DPT decoder **Cite**, and the architecture is shown in Figure **Blank**

Transformer models have a large amount of parameters. DepthAnythingV2 had **BLANK** many parameters, DepthPro has **BLANK**, and even the convolution-based **BLANK** models have **BLANK**. Even with regularization techniques within a model, like dropout and batch normalization **Cite**, and data augmentation techniques, like random cropping, color jitter, and random scaling, the "effective" number of parameters still dwarfs the number of training examples by several orders of magnitude. To avoid the issue of models overfitting to spurious correlations in the training data, which becomes

an even more significant issue when attempting to make large transformer models that generalize to unseen datasets, models must see a large amount of diverse data. This motivates the use of a synthetic focus stack generation pipeline, since we want to show our model an amount of data similar to the scale of training data used for recent MDE models.

2.4. Existing Datasets

The monocular depth estimation literature has several key pieces of prior work in regards to training data for new MDE models for both traditional MDE and DfD. Here, we will highlight a few datasets that were considered or are directly relevant for this paper. Hypersim is a diverse indoor synthetic dataset with more than 50,000 training samples. The ground-truth depth values of the Hypersim samples are log-normally distributed with a mean of 5.4 meters **Cite**. ARKitScenes is another indoor dataset with more than 1,000 unique scenes and more than 24,000 training samples. ARKitScenes is a real dataset with ground-truth depth labels collected using the LiDAR scanner of an iPad Pro, whereas Hypersim uses a synthetic computational pipeline to collect depth values of the scenes. TartanAir is a synthetic dataset with more than 250,000 training samples of indoor and outdoor scenes.

To our knowledge, DDFF 12-Scene is the only DfD specific training/evaluation dataset in the literature. It includes 600 training/evaluation samples captured through a lightfield camera in 6 unique scenes where each sample includes a 5-image focus stack with focus distances of 0.5, 0.63, 0.86, 1.33, and 7.06 meters respectively. [1] points out, and we concur, that this dataset is extremely challenging, to an extent that makes its challenges unrelated to the DfD task entirely. Many scenes in the dataset include textureless walls and tables, and the images were taken with a wide Depth of Field (DoF) such that there are barely any defocus cues in any of the focus stacks. To provide a better indication of real-world performance, prior DfD papers often synthetically blur higih-quality single image MDE datasets. For this purpose, we consider iBims-1, an indoor dataset with 100 AiF and ground-truth depth pairs. To use this dataset, as with other datasets like NYUDepthV2, prior DfD papers have synthetically blurred the images to obtain a focus stack and passed these into a

DfD model. **Cite NYUDepthV2 and other papers that have done this**

1 and 2 give an overview of the training and evaluation datasets considered and used for this paper.

3. Approach, 1-8, avg 3

The focus of this paper will be to show how we architected and trained a DfD model named **DfDNet** that beats the state-of-the-art on a couple of key evaluation datasets. We will go through the carefully selected training data for this task, pointing out key details regarding preprocessing that can accelerate future MDE work. The architecture of DfDNet was carefully constructed. It was motivated, first, by the DepthAnythingV2 model, and subsequent Video Depth Anything model. Both have a DINO encoder and DPT decoder transformer architecture, and the training paradigm of both models in their respective papers is that of large-scale training, indicating that we should also follow a similar training paradigm. Video Depth Anything specifically introduces a spatiotemporal head intended for temporal consistency across a set of up to 32 frames in a video at a time, but we choose to use that spatiotemporal head to create defocus aware features earlier in the encoder. We ablate the number of layers until the temporal collapse, the mechanism for the temporal collapse itself, as well as the use of training the model in a canonical, focal-length agnostic space.

Given the lack of training data for DfD, and the poor quality of the DDFF12 dataset that we noticed ourselves, and subsequently found to be pointed in other papers **Other paper**, we choose to synthetically generate focus stacks on the synthetic and real training and evaluation data. When looking for training data, we specifically look for large datasets (>5k images) with co-registered depth maps and AiF images. We also look for LIDAR-based datasets with relatively complete ground-truth depth maps and little sensor error.

In regards to evaluation data, we choose to evaluate on iBims with synthetic generation due to its high quality AiF images and depth maps, and we use DDFF12 as the only DfD evaluation dataset (though the quality is not very good)

Our hypothesis is that applying machine learning techniques of modern computer vision, like

transformers, large-scale synthetic data training, and broadly moving away from architectures that are derived from inductive biases will yield state-of-the-art results and model generalization. The main inductive bias that we do include in our model is moving the temporal collapse earlier since features should be defocus cue aware even after a couple of attention blocks, but we back this up with an ablation study.

To summarize our contributions, we:

- Introduce a new transformer-based DfD model called DfDNet (moving away from DfD being mostly convolution-based architectures trained on DDFF)
- Highlight high-quality datasets that can be used for training DfD models and specify preprocessing details necessary to make those datasets usable for DfD
- Evaluate on real and synthetic focus stack datasets to show that we beat the state-of-the-art both qualitiatively and quantitatively

4. Implementation, 10

4.1. Datasets Used

1 and 2 give an overview of the datasets considered for this paper. We chose to use Hypersim and ARKitScenes to train DfDNet for three main reasons. First, both datasets contain tens of thousands of training samples, which is on the appropriate order of magnitude for training a deep model. This first criteria also motivates excluding DDFF 12-Scene since it only has a few hundred training samples. The second reason for choosing Hypersim and ARKitScenes is that they both are indoor datasets with . Diagram BLANK shows the depth distributions of Hypersim and ARKitScenes, which shows that the mean depth of Hypersim and ARKitScenes are both less than 6 meters. On the other hand, TartanAir, as an outdoor dataset, is known to have large depth values as high as greater than 100 meters regularly in its training samples. Training on samples in an outdoor setting poses unique challenges due to the hyperfocal length phenomena (all images in the focus stack appearing like AiF images when objects in the image are sufficiently far away and the focus distance is also large), so we leave training on outdoor datasets to future work and also only evaluate in an indoor

setting. The last important criteria for the training datasets are the degree to which there are holes in the ground-truth depth maps. Hypersim has relatively few ground-truth depth map holes since there are only holes for windows or reflective surfaces. ARKitScenes has more holes due to sensor error given that ARKitScenes is a real-world dataset. In both of these cases, we filled the holes with a constant depth value of 1000. We knew we had to fill in the hole values somehow since the Gaussian convolution approximation of the CoC formula during synthetic focus stack generation involves applying a kernel over the entire image, so holes will propagate if we do not handle them carefully. We also noticed that since many of the holes were coming from ambiguous depth labels for the sky, we chose to label the holes with a correspondingly large depth value to distinguish those invalid values from the valid portions of the scene. However, since some reflective surfaces and arbitrary sections of the image were also given no depth value in the datasets due to sensor error and reflectivity, we applied a mask to the predicted depth of the model before applying a loss such that the model is not penalized for wrong depth values in sections of the image that were holes originally. Future work could fill in the holes using depth completion methodsf (cite Omni-DC, etc.) or explicitly filter out images with a sufficiently large portion of the image as invalid to provide a better heuristic.

Dataset	# of Samples	Depth Range	Diversity	GT Depth Holes	Data Type
Hypersim	66428	Indoors	✓	Few (sky+reflective)	AiF image
ARKitScenes	24647	Indoors	✓	Many (sensor error)	AiF image
DDFF 12-Scene	400	Indoors	✗	Many (sensor error)	Focus stack
TartanAir	279459	Indoors + Outdoors	✓	None	AiF image

Table 1: Training Datasets Considered

CHECK THESE, green for ones we chose and red for ones not chosen

4.2. Training Data Diversity and Augmentation

Obtaining diversity in the training data was a core motivating principle in training DfDNet. Hypersim and ARKitScenes themselves are both very diverse, both with over 1,000 distinct scenes in various indoor environments. To further augment the training data, we applied a random scaling/zooming

in of the images between 1.0 and 1.5. We also applied a 50/50 random flipping of the image horizontally and a random cropping of all images to have the input to the model being 480 x 640. We also applied a random color jitter to all images that randomly adjusts the brightness, contrast, and saturation of the images for the model to be robust to diverse real-world capturing settings. All of these previous data augmentation techniques are identical to the data augmentations done in OMNI-DC.

We deviated the training augmentation from prior literature in two main ways.

First, we randomized the F-number to be uniformly randomly selected from [10,11,12,13,14,18, infinity] in training settings where we intended to evaluate on the DDFF 12-Scene evaluation split since the DDFF 12-Scene dataset has an unknown exact F-number, but all that is known is that the DoF is high. When initially training in an F-number 1.4, 4, or 8 setting and evaluating on the DDFF 12-Evaluation set, we found that generalization to DDFF 12-Evaluation was low. This was believed to be because the models were overfitting to look for defocus cues specifically in the F-number setting that they were trained on, so randomizing the F-number in this way forces the model to learn to predict depth well in low defocus settings while also not overfitting to a specific F-number setting.

The second data augmentation we made was in regards to the focus distance sampling strategy during training. A heuristic for an ideal focus distance sampling strategy is one that generates a focus stack where the images defocus uniformly increases or decreases across the focus stack for any given point. The intuition for this is that a constant step increase or decrease in defocus across the focus stack allows a model to integrate defocus cues across the focus stack in a stable manner. Otherwise, the focus stack would either oversample images where there is limited new defocus information. Consider the following derivation from the CoC formula **ref it**:

$$
\begin{aligned}
\text{CoC} &= \frac{|d - d_f| \cdot f^2}{2 \cdot a \cdot d \cdot (d_f - f)} \\
&\overset{d_f - f \to d_f \text{ when } f \ll d_f}{\approx} \frac{|d - d_f| \cdot f^2}{2 \cdot a \cdot d \cdot d_f} \\
&= (\frac{d}{d \cdot d_f} - \frac{d_f}{d \cdot d_f}) \cdot \frac{f^2}{2 \cdot a} \\
&= (\frac{1}{d_f} - \frac{1}{d}) \cdot \frac{f^2}{2 \cdot a} \\
\text{CoC} &\propto (\frac{1}{d_f} - \frac{1}{d})
\end{aligned}
\tag{2}
$$

Thus, for a fixed depth d, a uniform step in inverse focus distance (d_f) space is approximately a uniform step in defocus space. This approximation breaks down for very small focus distances since we relied on the fact that the focal length is usually around 50 millimeters, whereas focus distance is in meters. Thus, we sample focus distances uniformly in inverse depth, or disparity, space. However, we still have to identify the lower and upper bound in disparity space to interpolate between. We could identify the 5th percentile and 95th percentile ground-truth disparity in the training setting for the lower and upper bound to linearly interpolate between for the sampled focus distances to synthetically generate, but this means that the focus distances themselves serve as a prior that encodes the scale of the scene. In both real-world and evaluation settings, the focus distances sampled in DfD is agnostic to the scene depth, so we instead choose to randomly sample the lower and upper bound for the focus distances. We decided to have $\text{disp}_{\text{lower}} \sim Unif(0.05, 0.25)$ and $\text{disp}_{\text{upper}} \sim Unif(1, 2)$. The disparity lower bound being uniform between 0.05 and 0.25 means that the maximum sampled focus distance can be anywhere between 4 and 20 meters, which covers the range of indoor scenes. The disparity upper bound being between 1 and 2 means that the depth lower bound for the sampled focus distances is between 0.5 and 1 meter, which seemed to capture the distance of the closest object to the camera in indoor scenes in both the training and evaluation datasets. A potential drawback of this focus distance sampling strategy is that some generated focus stacks do not have any useful defocus cues. For example, an indoor scene where the closest object is actually 4 meters away from the camera but the $\text{disp}_{\text{lower}} = 0.25$ would have all the sampled

focus distances be closer than the closest object in the scene, meaning the entire scene appears out of focus in the entire focus stack. Future work could propose a more sophisticated focus distance sampling strategy that balances scene-agnostic depth sampling to make training appropriately hard while also avoiding afformentioned training samples that are unnecessarily hard. Ideally, this more sophisticated focus distance sampling strategy could be loaded onto a real camera such that focus stacks can be collected programatically in real-world settings, though the sampling strategy we propose here, combined with a model that detects if the focus stack has sufficient defocus cues and resamples the focus stack if not, is probably sufficient.

4.3. DfDNet

Here, we introduce the architecture of DfDNet, the first transformer-based DfD model. DfDNet has an encoder-decoder architecture similar to that of DepthAnythingV2. The most interesting differences are within the encoder, which is derived from the well-known DINOv2 encoder with two main differences. In the following description, we use "temporal" dimension and "focus stack" dimension interchangeably. First, we apply a temporal attention module after every spatial attention block for the first m attention blocks. After the $m+1$-th DINO spatial attention block, we include an average pooling layer that "collapses" the features across the focus stack dimension. The motivation for this is twofold. First, a hypothesis we had was that the model can encode all the defocus information it needs before the 12th temporal attention module. The second reason is that collapsing the focus stack reduces the number of features being passed through the model by a multiple of n, where n is the size of the focus stack ($n = 4$ in the diagram, and the batch size is 2). This allows us to run a faster inference, speedup training, and generally be more memory efficient. We ablate the exact number of layers before the collapse that is optimal, the use of average pooling versus max pooling, and the inference time comparisons in the evaluation section.

The temporal attention module itself has a few interesting features to note. The module is motivated and derived from the "motion module" and spatiotemporal head introduced by the authors of Video Depth Anything. There are also other video depth papers that introduce similar temporal

attention modules (**Cite FlashDepth, etc.**), but we chose to write code on top of Video Depth Anything's released motion module code since Video Depth Anything shares an architecture very similar to DepthAnythingV2 already, so migrating the Video Depth Anything motion module code to DfDNet seemed to be easier. Even though video depth estimation (VDE) seems to be a different problem than DfD, our core realization was that the problems are actually very similar. Of course, they both are monocular depth, but they both also have the core requirement of needing to integrate information across some type of temporal dimension (focus stack for DfD and frames over seconds for VDE). There are, however, a few key differences. First, VDE involves m input frames and m corresponding depth prediction frames as output, whereas DfD is single-view and has m input frames but only 1 output depth prediction frame, which is why we must do some type of pooling to collapse the temporal dimension. Another important difference is the embedding used by Video Depth Anything's motion module versus DfDNet's temporal attention module. Since attention is permutation invariant with respect to the input features, Video Depth Anything had to explicitly embed the order of the frames via a positional embedding to leverage the temporal consistency signal for depth prediction on a video. However, DfD does not leverage temporal consistency merely from the order of the input frames, but rather from the focus distances associated with each frame. Since the focus distance input is 1 metric number for each frame in the focus stack, we passed the focus distances into an MLP that maps each 1-dimensional FD to be the same dimensionality as the focus stack features. Then we added the embedded focus distance vector to the focus stack features since the dimensionalities match up and used this new set of FD+Focus Stack feature vectors as the input for a self-attention block that involves isolating the temporal dimension from the batch dimension and attending over the temporal attention for each feature in the focus stack. The result of this is a set of FD-aware, temporally attended set of features, and as the functions f, g, and h that map the FD+focus stack features to the query, key, and value vectors respectively are learned along with the FD embedding MLP, this module can support the integration of defocus cues into the focus stack features.

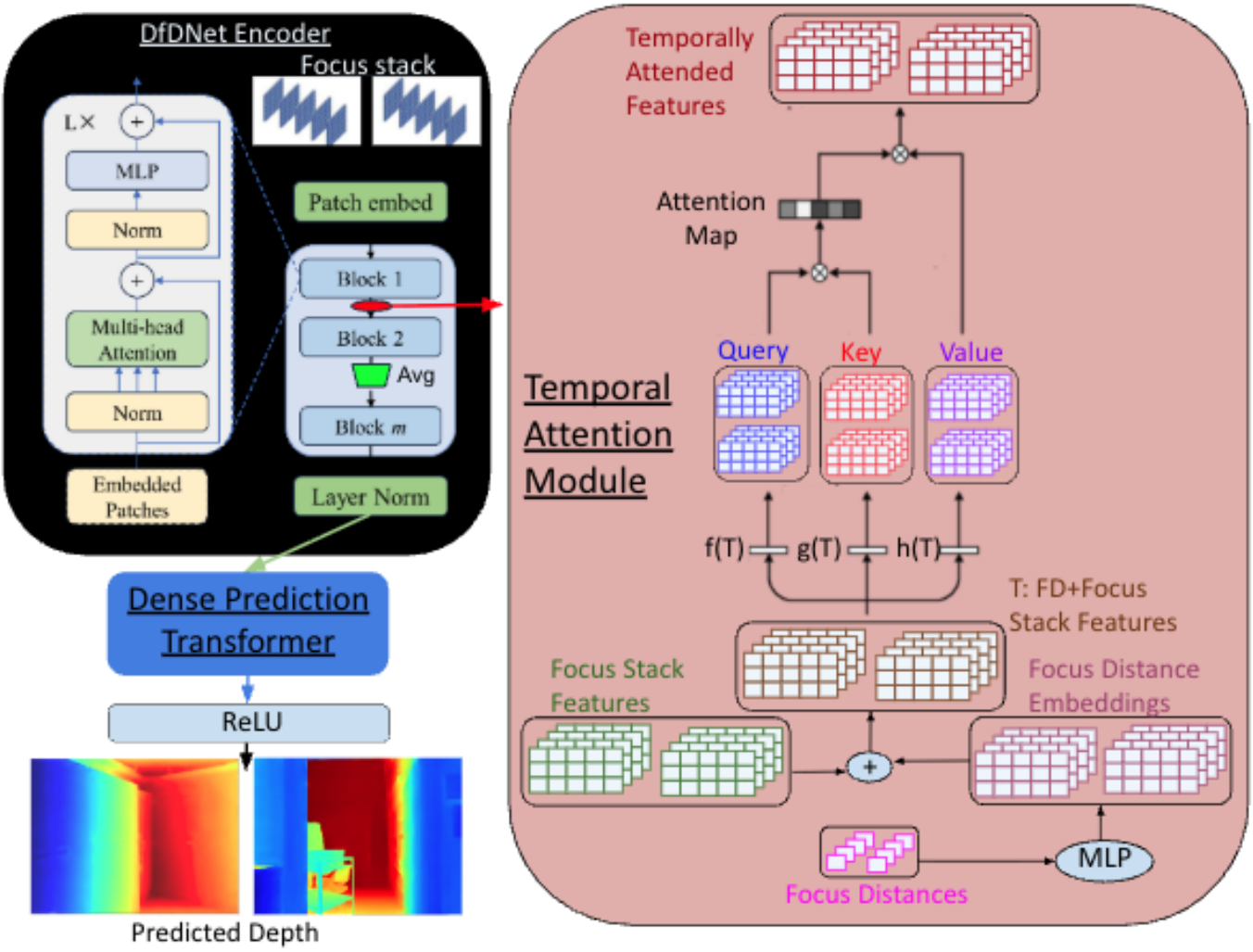

Figure 3: This is a gray image.

4.4. Canonicalization

DfDNet-canonical indicates that we also canonicalized the focus distances and target depth when training to have the model learn to predict depth in a canonical space. The scaling factor for canonicalization is $\frac{w}{max(f_x,f_y)}$, where w is the width of the image and f_x and f_y are the horizontal and vertical focal length of the camera respectively. The motivation for this comes naturally from the fact that the projection of a 3D point (X,Y,Z) onto the image plane to the 2D point (x,y), assuming $f_x \approx f_y = f$, follows the formula $x = f\frac{X}{Z}$ and $y = f\frac{Y}{Z}$. Rearranging, we get $\frac{x}{f} = \frac{X}{Z} = \tan(\theta_x)$ and $\frac{y}{f} = \frac{Y}{Z} = \tan(\theta_y)$, where θ_x and θ_y correspond to the horizontal displacement angle and vertical displacement angle, respectively, of the ray extending from the principal point

of the camera to (X,Y,Z). By dividing the target depth and focus distances during training by $\frac{w}{f}$, where $max(f_x, f_y) = f$, we normalize the correspondence between the 2D image coordinate and the ray to the 3D point.

At evaluation time, we undo the canonicalization by the same scaling factor. This procedure is identical to the canonicalization procedure in DepthPro and Metric3D.

4.5. Loss

The training objective for DfDNet has two terms: SiLog and gradient matching scale, with a tunable hyperparameter α to balance them.

$$\mathcal{L} = \alpha \mathcal{L}_{\text{silog}} + \beta \mathcal{L}_{\text{gmscale}} \tag{3}$$

The first term is a balance between a scale-invariant log and ℓ_2 loss term:

$$\mathcal{L}_{\text{silog}} = \sqrt{\frac{\sum_i d_i^2}{\ell} - \lambda (\frac{\sum_i d_i}{\ell})^2} \tag{4}$$

where $d_i = \log y_i - \log \hat{y}_i$ and y_i is the target depth and $\hat{y}_i$ is the predicted depth over only the valid pixels (non-masked). ℓ is the number of valid pixels and $\lambda \in [0,1]$ but is usually set to 0.5 in the literature, so we do the same in this paper.

The second term is a gradient matching scale loss.

$$\mathcal{L}_{\text{gmscale}} = \frac{1}{n} \sum_k \sum_i |\nabla_x R_i^k| + |\nabla_y R_i^k| \tag{5}$$

where $\nabla_x R_i^k$ is the difference between the prediction and target in the x direction at scale k of the gradient between neighboring valid pixels. $\nabla_y R_i^k$ is defined similarly in the y direction. n is the number of valid pixels, and we sum over $k = 0,1,2,3$ to capture different scales over larger distances in the input image.

Here, α and β are both tunable hyperparameters for balancing both loss terms.

5. Evaluation, 5.5

Dataset	# of Samples	Depth Range	Diversity	GT Depth Holes
iBims-1	100	Indoors	✓	Few (sky+reflective)
DDFF 12-Eval	200	Indoors	✗	Many (sensor error)

Table 2: Evaluation Datasets

Notes:

- Experiments that were run
- iBims-1
- DDFF12
- Chart showing results
- Table showing results of different modelsCommentary on evaluation data quality
- Inference time chart vs model size
- Inference time vs iBims/DDFF accuracy
- Ablation: number of layers until collapse/temporal fuse method and iBims performance – 24-32
- Ablation: With and without canonicalization performance 49-96
- Ablation: with and without arkit performance/with and without hypersim performance 49-96
- Ablation: FD sampling strategy

6. Conclusions, 1.25

-

References

[1] M. Maximov, K. Galim, and L. Leal-Taixe, "Focus on defocus: bridging the synthetic to real domain gap for depth estimation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, ser. CVPR '20. USA: IEEE Computer Society, 2020.