Group 7 - Monocular Depth Estimation

Antonio Calderoni Giovanni De Muri Elisa Degara Benedetta Zanini

1 Introduction

In this project, we deal with the task of Monocular Depth Estimation. Monocular Depth Estimation is a computer vision task, consisting in estimating the depth value, that is the distance relative to the camera of each pixel, given a single (monocular) RGB image. The task has many real-world applications, ranging from 3D reconstruction to augmented reality and robotics. For instance, in autonomous driving it is crucial to understand the position of objects that are out of reach for sensors. Differently from other types of depth estimation tasks, here we are only using one image. This can be useful for settings in which stereo images data are not available, or in which computational resources are limited.

Thus, the input for this task is a single RGB image, and the output is a depth mask, an image of the same size as the input, but with only one channel, where each pixel represents the relative distance from the camera of the corresponding pixel in the RGB image; you can see an example in Figure 1.

Figure 1: Image and corresponding depth mask

The goal of this project is to explore different architectures that can be used for the task, and analyze which one performs better. We are going to focus on CNN-based architectures.

2 Related work

To build effective models, we took inspiration from previous research in the field. In particular, we focused on works that implement Convolutional Neural Networks (CNNs) models for depth estimation tasks.
As pointed out in the work "Single Image Depth Estimation: An Overview" by Mertan et al., CNNs are effective at extracting depth information from single images and there have been different ways in which they have been employed for such a task.
The most straightforward approach is an **encoder-decoder** architecture. The encoder section of the network performs the function of a feature extractor by examining the RGB image input and obtaining insightful representations at various abstraction levels. These features are able to encode important elements of the image, such as edges, the geometry of the space and object interactions.

It does so by progressively decreasing the input image resolution, using convolutions and pooling operations. This allows to increase the receptive fields at each layer, thus capturing more global information. Using the associations it has learnt, then, the decoder part takes the encoded features and gradually upsamples them to produce a dense depth map with the same resolution as the input image. The network is able to retrieve spatial information through this upsampling process, that converts the encoded characteristics into a useful depth prediction for every pixel.
As for the encoder part, often pretrained models are used, such as Residual Networks (ResNets) and Densely Connected Convolutional Networks (DenseNets). This can help since these models have already learnt to efficiently extract high-quality features. For instance, the architecture implemented in the paper "Deeper Depth Prediction with Fully Convolutional Residual Networks" by Laina et al. consists of an encoder-decoder network, using ResNet-50 for the initial portion of the network, which is initialized with pre-trained weights. With a series of unpooling and convolutional layers, the decoder part of the architecture up-scales the output of the first, and allows to get a prediction.
Another type of architecture that is often used is **UNet**. Similarly to the previous one, the architecture consists of an encoder and a decoder, with the difference that there are skip connections that connect layers of the encoder and the decoder, implemented through concatenation. This implementation is more suitable to retain the spatial information of the image, along with the information from the features, that otherwise could be lost during downsampling.
As an example, in the paper "High Quality Monocular Depth Estimation via Transfer Learning" by Alhashim et al. the authors built a UNet architecture, using as the encoder the pre-trained DenseNet-169 network, which is truncated to remove the top layers related to the classification task. The decoder then consists of a series of upsampling blocks and convolutions, along with the concatenations from the decoder's layers. A leaky ReLU activation is applied after each upsampling block, except for the last one.

3 Data

In our project, we employed the NYU-Depth V2 dataset, which can be found at this link. The dataset consists of indoor scenes images, such as bedrooms, kitchens, study rooms, with around 1449 densely labeled pairs of RGB and depth images and over 400,000 unlabeled frames, used mostly for unsupervised learning tasks.
The labeled data include pre-processing steps like filling in missing depth values to ensure consistent training. Both the RGB images and the depth maps in the dataset are 640 x 480 pixels in resolution. For our project, we decided to focus on the labeled images, obtaining a training set that consists of 1111 images and a test set of 338 images. We decided to use data augmentation on the training set in order to increase the variability of the dataset, thus improving the robustness of the model. For computational reasons, moreover, we decided to downsize the images to 128x128.

In particular, we augmented the training data using random transformations:

- Random Rotation: Both the RGB image and the depth map are rotated by $r \in [-5, 5]$ degrees.
- Random Crop: Both the RGB image and the depth map are randomly cropped to 480x480 pixels.
- Random Color Jitter: The brightness of the RGB image is multiplied by $c \in [0.8, 1.2]$
- Random Horizontal Flip: The RGB image and the depth map are horizontally flipped with probability 0.5.

Note that these transformation preserve the geometry of the scene. The test data, instead, was just rescaled, and no data augmentation was performed. We have also decided to normalize the depth values between 0 and 1, in order to possibly stabilize training, and better understand the loss values.

4 Metrics and Loss Used

The choice of a suitable loss is very important in depth estimation, since it can have a big difference on the final depth map produced.

First of all, depth estimation is an inherently ambiguous task. Indeed, given a 2D image, there are an infinite number of possible 3D scenes that may have produced it. Thus there can be various possible depth maps for a given image. This ambiguity can be usually resolved since most scenes are physically implausible. However, one important ambiguity that remains is the global scale. Thus, what we are looking at is a relative scale, since we have no absolute reference of depth of the scene, since we are using just one image. As for the loss, this problem is usually addressed using scale-invariant losses, that focus more on spatial relations within a scene rather than the global scale. So in this case the focus is the relative depth: we care about the relative ordering of the pixels, instead of the absolute value. In this way the system is not penalized for mistakes due to scale ambiguity.
Moreover, in depth estimation tasks sometimes the errors are computed in log space. This is due to the fact that, even if we make a small mistake of the depth when the object is far away, we will not really care about it. However, if the object is closer, them we might want to be more precise in our prediction.
There is no universal loss that is used in the field, but some choices that could be made are the L1 Loss (it is more robust to outliers and it promotes sparsity), RMSE (it penalizes larger errors more heavily and it tends to produce blurry results), the SSIM (it focuses on the structural similarities of the images; thus it tries to preserve the perceptual quality of the depth), the Gradient Loss (that penalizes differences in the gradients of the depths, helping preserve the local structure), and the BerHu (it acts as an L1 loss below a threshold and as a L2 above it, thus penalizing more heavily big mistakes, but also accounting linearly for small ones). Often they are combined in order to produce a loss that takes into account various aspects of the predicted depth. We focused mainly on the L1 Loss due to the robustness to outliers (since we observe that the labeled training data sometimes contains noise), and since empirically we observed that it performs well.
Considering that we are not dealing with classification, it's also important to specify what metrics we will use to evaluate different architectures. There is no universal metric in depth estimation tasks, but we will use mainly the RMSE (L2) and absolute relative error (L1) loss. You can find various benchmarks and state-of-the-art models at this link. It is important to point out that we did not test our models on data used for benchmarks, so our loss values are not comparable to those you can find at the link.

5 Methodology

In this section, we are going to discuss the approaches and architectures we used to tackle the problem. We will discuss 6 models that we have built. As already mentioned, we initially focused on architectures that have an encoder-decoder structure, which initially decrease the image resolution, increasing the number of features, and then upsample the result in order to obtain a single depth image. We chose to use already pretrained models as the encoder since, overall, it can lead to better performance, due to the fact that they have already learned an high-level representation of the input. Moreover, pretrained weights can facilitate convergence.
The first model **DE NoSkip** we built is a simple encoder-decoder model, without skip connections between the two parts of the network. As for the encoder part, we decided to use a pretrained ResNet50, a model initially trained on ImageNet, that uses residual connections with convolutional blocks in order to obtain a downsampled tensor with 2048 channels. We decided to remove the classification head of the original architecture, along with the last convolutional block. This last step was justified by the fact that, empirically, the model performed better. Since we are using 128x128 images, using that many layers reduced the size of the feature to 4x4, and this may cause performance to worsen. Between the encoder and the decoder, we inserted a convolutional layer followed by BatchNorm for regularization. As for the decoder part, we decided to use blocks made of a bilinear upsampling layer (it increases the image size by using nearby pixels to compute the pixel value, through a linear interpolation. The downside of this method is that it can generate blurry results), and a convolutional layer. Each of them is followed by a ReLU activation function. We stacked together 4 such blocks, using 3 skip connections. We did so by implementing a convolutional

layer and an upsample layer from the lower resolution feature map. After each connection we applied the batch norm regularization. As for the output layer, we just used a simple convolutional layer, followed by the ReLU activation.

We then built upon the last architecture, with some modifications, to implement the skip connection between the encoder and the decoder, typical of the UNet architecture. In particular, we first tried using the sum, as in a normal skip connections, and then we tried the concatenation. As we already discussed, we expect the latter to work better since they better retain the spatial information of the initial image. We also tried different number of concatenations at different layers, in order to see how this affected the final result.
The first UNet-inspired model **ResNetE Decoder v1** uses ResNet34 as the encoder. ResNet34 is just a smaller version of ResNet50, in which the convolutional block is changed a bit. In this case, using 128x128 images, the encoder bring the features down to 4x4, and it uses 512 channels. However, thanks to the skip connection, we tried using the full ResNet34, without removing the first convolutional block. Nonetheless, we removed the classifier head. The decoder is composed of five blocks, where each blocks is made of an upconvolutional layer (we used a transposed convolution operator, that can be learned. While more computationally expensive than bilinear upsampling, we decided to try it since we are using a smaller encoder) and a convolutional layer, both followed by a ReLU activation function. The last block is a sequence of convolutional layer. Here, we introduced two types of skip connections, both implemented using the sum, followed by a batch norm regularization layer. There is one skip-connections after each block of the encoder, connected to each block of the decoder. These connections are typical of the UNet architecture. The other are residual connections inside the decoder itself, and there are only 3 of them. This last type of skip connection is implemented, as in the previous model, using a convolutional layer followed by bilinear upsampling, and a ReLU activation. The last layer is instead a convolutional layer, followed by a final upsample.
The second UNet-inspired model, **ResNetE Decoder v2**, is similar to the previous one. However, some of the skip connections introduced before are modified. Indeed, in this architecture we removed all the skip connections in the decoder, except for the one coming from the last block, while we left unchanged those coming from the encoder. Moreover, we added a skip connection coming directly from the input. The input, the skip connection coming from the decoder, and the output of the network are concatenated along the channel dimension at the end, immediately before a final convolution. This allows us to exploit the ability of the concatenation to retain spatial information.
Lastly, we build another model **ResNetE Decoder v3**, with the idea of simplifying the previous two networks, and modifying the skip connections. This network still has a ResNet34 encoder, with skip connections to the decoder. The difference here is that all skip connections are implemented through concatenation along the channel dimension. The skip connection coming from the input is kept, as we thought that it may help retaining geometric features. The decoder block is simpler than the one of the previous models, adopting only 2D Transposed Convolution, followed by a ReLU activation function. The last block is made by three normal Convolutional layers. Once we built these models, we wanted to see the performance of UNet using different and possibly bigger models. In particular, we tried Dense Net 121 - a CNN architecture with 121 layers, characterized by blocks where each layer communicates with every other layer in a feed-forward way - and ResNet50. The main difference between using the two encoders is that with ResNet we obtain 2048 channels, while with DenseNet, instead, we obtain 1024 channels. This difference in the numbers of output channels captures a different feature representation of the models, and this could have an effect on the final depth map.
In the model **DenseNet Unet**, we used the pretrained DenseNet121 as the encoder. After the encoder, an intermediate convolutional layer is applied and then the extracted features are passed to the decoder. The decoder is made of four blocks. Within each block, we applied bilinear upsampling to double the spatial dimensions of the input. This last process is followed by concatenation operations, where the upsampled feature maps are combined with the corresponding feature maps from the encoder. Finally, in each block we include also two convolutional layers with Batch Normalization and ReLU activation, to refine the feature representations. At the end of the decoder, we apply a final convolutional layer to reduce the number of channels to 1 and produce the final depth mask.
The model **ResNet UNet** follows the same approach, but we opted for ResNet50 instead of using DenseNet121 as the pretrained encoder. Differently from the simple encoder-decoder structure

used before, here we are using the full ResNet50, without the classification head. We expect that the concatenation with the encoder allows to keep the spatial information, even though the features map becomes very small.
After having built these models, we experimented with different losses. In particular, we tried using the L1 Loss, the BerHU loss, and a custom loss, that used both the SSIM and the L1 loss with equal weight. We used ResNet UNet as a benchmark model to analyze the loss. Both visually and studying the metrics, the L1 Loss was the one that performed the best when training the model for 50 epochs.
We then trained each models for 50 epochs. We used a batch size of 64, and we used Adam as optimizer. We tried experimenting also with weight decay, but it did not bring any improvement to the model; on the other hand, we may even say that it brought negative changes. The performance metrics are summarized in the table in Figure 2, while the prediction on some samples can be seen in Figure 3.

Model name	L1 Error on test data	MSE on test data
ResNetE Decoder v1 (50)	0.0939	0.0194
ResNetE Decoder v2 (25)	0.0976	0.0190
ResNetE Decoder v3 (50)	0.1006	0.0207
ResNet UNet L1 loss (50)	**0.0845**	**0.0171**
ResNet UNet BerHu loss (50)	0.0878	0.0175
ResNet UNet Custom loss (50)	0.1231	0.0299
DE NoSkip (50)	0.0891	0.0179
DenseNet UNet (100)	0.0972	0.0234

Figure 2: Models' performances

6 Results

Analyzing the result of the training for 50 epochs of the DE NoSkip model using the L1 Loss, we observe that the model manages to estimate the depth of big regions almost correctly, as confirmed also by a low L1 and L2 test loss. However, the details of the images are lost, and the prediction is very blurry. This is what we could have expected, due to the the fact that architectures produce lots of high-level features, but loses the spatial information and the details of the initial image due to the downsampling. During training, moreover, we observe the following phenomenon. Although both the train and the test loss went down, the predictions did not change much and were almost identical during training. This problem might be due to the loss not being able to fully capture bad predictions. Indeed, the L1 loss does not take into account the spatial structure of the image, and thus does not help the model in learning edges.
When training the initial version of the ResNet-Encoder Decoder model (ResNetE Decoder v1) using L1 loss for 50 epochs, we also observed that the predictions were quite blurry. Although the predicted depth maps represents regions of high and low depth correctly, the resulting predictions do not allow for the recognition of distinct objects. Thus we deduce that the model effectively captures the relative distances between areas in the image, but it does not retain precise geometrical shapes.
Contrarily, outputs of ResNetE Decoder v2 perfectly encode the object geometry at the expense of a reduced understanding of the depth. The sharp border recognition is much likely given by the concatenation as second-to-last step with the input images. Indeed, model v2 is able to recognise what are the closest elements without retaining information of the surrounding objects depth. While training v2 with L1 loss, we also noticed that after 20/30 epochs the test loss started to oscillate, without decreasing anymore, although the training loss still decreased. This was the case in some of the models we trained, and is probably given by the small size of the train dataset, of

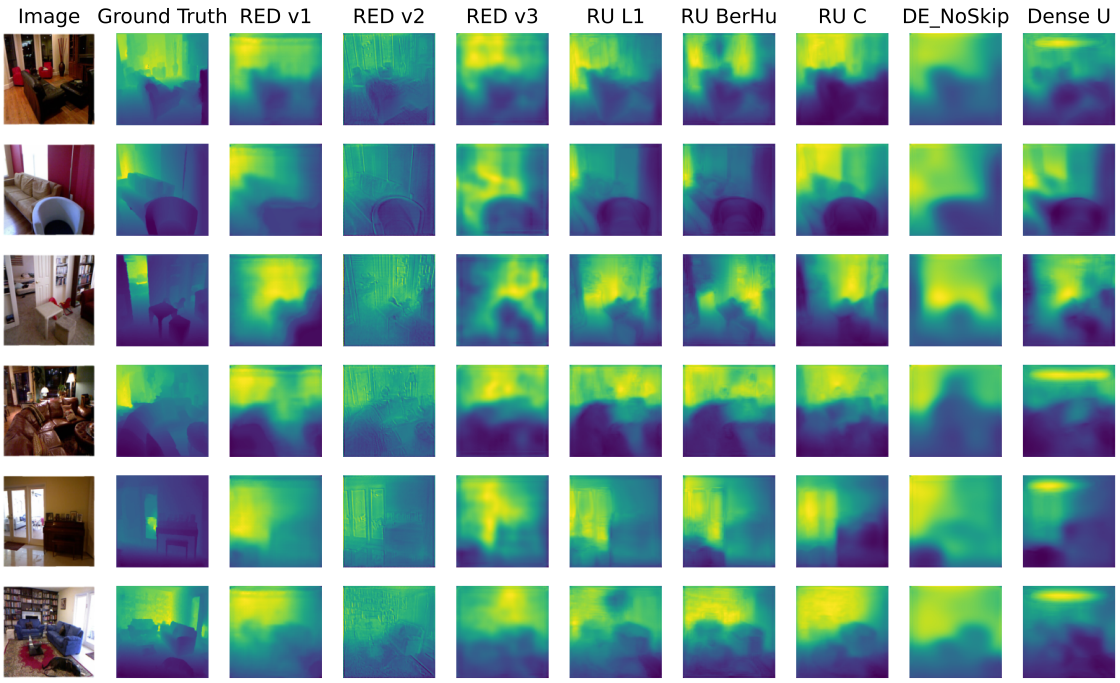

Figure 3: Models' predictions

around 1000 images.
The last ResNetE Decoder model (v3) outputs predictions that combine aspects of both v1 and v2 predictions. It retains less geometric information than v2 but more than v1. Additionally, it recognizes the depth of different regions better than v2, but not as well as v1. The L1 error and Mean Squared Error on the test data are slightly worse in v3 compared to both v1 and v2; however, to the human high the prediction of this model are better. Thus, we have to be careful when just looking at the L1 and L2 losses, since they may not be the best metrics for the task. Indeed, this can also be seen in the DE NoSkip model, in which the predictions are visually bad and blurry, but the test loss is among the lowest.
As for DenseNet UNet model, we observe that it manages to keep some details of the images, although they are less clear and defined than other models. As for the depth, it is not very precise, even though it manages to learn it broadly. For instance, among the tested model, it is the one with the highest L2 loss on the test set. Indeed, we see that there are locations where the depth map is quite off from the truth. Visually, this can be observed in 3: near the top of the image the depth is often predicted further away than the real prediction.
The last and best performing model is the ResNet UNet trained for 50 epochs using L1 loss, achieving an L1 error of 0.0845 and an MSE of 0.0171. Visually, the predicted depth maps closely resemble the ground truth. Interestingly, although the ResNet UNet trained with BerHu loss has worse performance metrics, its outputs appear slightly better to the human eye. Indeed, this loss helps in preserving details, since it more strongly emphasizes larger errors. Instead, adding the SSIM loss to the L1 loss did not seem to improve the accuracy of the depth map. In general, the model behaves well using the different losses and manages to keep spatial information from the initial images. Moreover, we observe that it gives better prediction than Dense UNet, even though the network is very similar. Thus, it seems that the decoder is able to better interpret the features from ResNet.

7 Further improvements

As previously mentioned, monocular depth estimation is a non trivial task because an infinite number of depth maps can correspond to a single image. The correctness of a depth map is determined by the relative predicted depth between two elements in the image. When building a model for depth estimation, one must balance penalizing errors in depth estimation of large

areas with errors related to incorrect object segmentation. For this sake, the loss choice is crucial, therefore as a potential improvement, Moreover, assessing a model's performance in this task requires more than just looking at performance metrics. As shown in previous sections, some models may produce visually worse predictions despite having promising scores. For instance, the DE NoSkip model generates visually poor predictions compared to the ResNetE Decoder v3, yet it has better scores. Another observation is that after approximately 30 training epochs, the models we trained showed no improvement in test set performance, even though the training loss continued to decrease. This suggests that the models were not learning effectively beyond this point. As a potential improvement, using a larger training dataset could help enhance model performance.