High Quality Monocular Depth Estimation via Transfer Learning

Bianca-Alexandra Ghiorghiu, Maria-Ecaterina Constantinescu, George-Vlad Enache

***Abstract*—The objectives of this project include replicating the original steps to build a monocular depth estimation model using DenseNet169, implementing an alternative model using DenseNet121, training both models on the same dataset, and comparing their performance qualitatively and quantitatively. To achieve these objectives, visualizations of depth maps and input images will be used to compare the qualitative results of both models. Additionally, the proposed evaluation metrics such as root mean squared error, average relative error, average logarithmic error, and threshold accuracy will be employed to assess the quantitative performance of the models. The results will be analyzed to draw conclusions on the effectiveness of DenseNet121 compared to DenseNet169 in the context of monocular depth estimation via transfer learning.**

***Index Terms*—DenseNet169, DenseNet121, Depth Estimation, Transfer Learning, Encoder-Decoder**

I. Introduction

THE aim of this project is to replicate the steps outlined in the paper "High Quality Monocular Depth Estimation via Transfer Learning" by Ibraheem Alhashim and Peter Wonka, and further investigate the impact of using DenseNet121 instead of DenseNet169 in the architecture. The project will focus on implementing the monocular depth estimation model and comparing the performance of DenseNet121 and DenseNet169, both qualitatively and quantitatively, using the evaluation metrics proposed in the original research.

A. Objectives

- Replicate the steps in "High Quality Monocular Depth Estimation via Transfer Learning" to build a monocular depth estimation model using DenseNet169.
- Implement a monocular depth estimation model using DenseNet121 as an alternative to DenseNet169.
- Train both models on the same dataset to ensure a fair comparison.
- Compare the qualitative results of both models using visualizations of depth maps and input images.
- Evaluate the quantitative performance of both models using the proposed evaluation metrics, such as root mean squared error, average relative error, average logarithmic error and threshold accuracy.
- Analyze the results and draw conclusions on the effectiveness of DenseNet121 compared to DenseNet169 in the context of monocular depth estimation via transfer learning.

B. Methodology

- Review the original paper and understand the architecture, training methodology, and evaluation metrics used in the research.
- Collect and preprocess the dataset used in the original paper, ensuring consistency in data quality and format.
- Implement the monocular depth estimation model using DenseNet169, following the steps outlined in the original paper.
- Train the DenseNet169 model using the dataset and document the training results.
- Modify the architecture to incorporate DenseNet121 as the backbone and implement the monocular depth estimation model.
- Train the DenseNet121 model using the same dataset and document the training results.
- Compare the qualitative performance of both models by visualizing depth maps and input images.
- Evaluate the quantitative performance of both models using the evaluation metrics proposed in the original research.
- Analyze the results to draw conclusions and provide recommendations on the choice of backbone architecture for monocular depth estimation via transfer learning.

II. Theoretical Background

A. Model Architecture

For the architecture, the paper proposes an encoder-decoder network using a pre-trained DenseNet169 for the encoder and a series of up-sampling layers with skip connections for the decoder. The encoder is the part of the model that processes the input data and extracts important features while the decoder refers to the part of the network that takes the encoded features generated by the encoder and transforms them back into the original input size. As such, the overall architecture can be summarized as presented in 1.

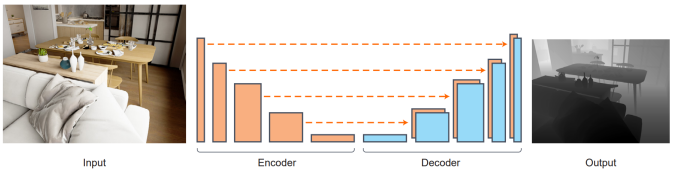

Fig. 1: Model architecture [2]

1) Encoder: The DenseNet169 architecture consists of up of several dense blocks, each of which is made up of a stack of densely connected convolutional layers and transition layers that downsample the feature maps. The general architecture presented in the figure 2 can be described as follows [6]:

The Input Layer: The input of the network consists of a 224x224x3 RGB image.

Initial Convolution: The input image is passed through a single convolutional layer with 64 filters and a kernel size of 7x7, followed by a max pooling layer with a stride of 2.

Dense Blocks: The network has four dense blocks, each with several dense layers and a growth rate of 32. Each dense layer concatenates the input with the output of all preceding layers in the block, and the resulting feature map is fed into a series of convolutional layers.

Transition Layers: Between each pair of dense blocks, there is a transition layer that performs downsampling of the feature maps. The transition layer consists of a batch normalization layer, a 1x1 convolutional layer with 32 filters, and a max pooling layer with a stride of 2. The purpose of the transition layer is to reduce the dimensionality of the feature maps while maintaining their representational power.

Global Pooling and Classification: After the final dense block, there is a global average pooling layer that averages the feature maps across the spatial dimensions. The output of the global pooling layer is fed into a fully connected layer with 1000 units, followed by a softmax activation function that produces the class probabilities.

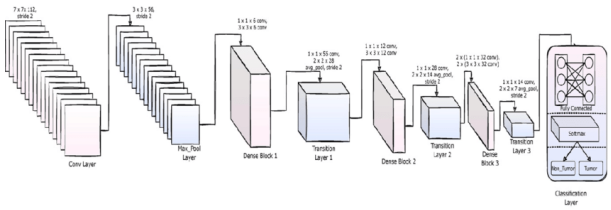

Fig. 2: DenseNet169 architecture [1]

The DenseNet121 architecture consists of the same blocks as DenseNet169, the main difference between DenseNet169 and DenseNet121 being the depth of the network and the number of parameters. DenseNet169 is a deeper architecture than DenseNet121, with 169 layers with a larger number of convolutional filters in each layer and a total of 14.3 million parameters compared to 121 layers and 8 million parameters.

2) Decoder: The architecture of the decoder presented in the paper is described as follows [2]:

The first layer consists of a 1 × 1 convolutional layer with the same number of output channels as the output of the truncated encoder. This layer is used to increase the number of channels in the encoded features to match the number of channels in the original input image.

A series of upsampling blocks are added to the decoder. Each upsampling block starts with a 2× bilinear upsampling operation, which doubles the spatial dimensions of the input feature maps.

The upsampling blocks are followed by two 3 × 3 convolutional layers, each with output filters set to half the number of input filters. The first convolutional layer of the two is applied to the concatenation of the output of the previous layer and the pooling layer from the encoder having the same spatial dimension.

Each upsampling block, except for the last one, is followed by a leaky ReLU activation function with parameter $\alpha = 0.2$. This helps to introduce non-linearity into the decoder and prevent the vanishing gradient problem during training.

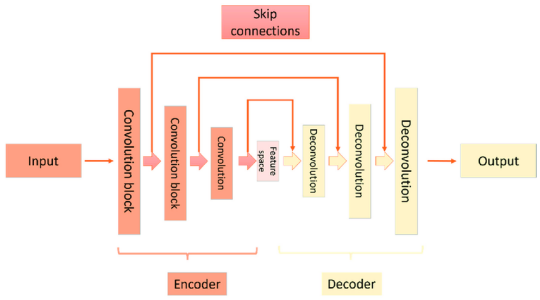

Fig. 3: Encoder-Decoder architecture with skip connections [3]

B. Loss

The authors' goal with their method is to create a loss function that strikes a balance between reconstructing depth pictures by reducing the difference of depth values and punishing distortions of high frequency features in the depth map's image domain. These high-frequency features often match scene object borders. The suggested loss function encourages the model to create depth maps that are not only correct in terms of depth values but also visually consistent and cohesive with the underlying scene structure by punishing distortions of these aspects.

For training our network, the loss L is defined as the weighted sum of three loss functions between the groundtruth depth map y and the predicted depth map $\hat{y}$ [2]:

$$L(y, \hat{y}) = \lambda L_{depth}(y, \hat{y}) + L_{grad}(y, \hat{y}) + L_{SSIM}(y, \hat{y}) \quad (1)$$

The depth loss function, L_{depth}, calculates the mean absolute error between the predicted depth map and the ground truth depth map. It is defined as:

$$L_{depth}(y, \hat{y}) = \frac{1}{n} \sum_{p}^{n} ||y_p - \hat{y_p}||_1 \quad (2)$$

The gradient loss function, L_{grad}, measures the difference in gradient magnitude between the predicted depth map and the groundtruth depth map. This loss function penalizes the network for generating depth maps that have sharp transitions or edges in areas where the groundtruth depth map has smooth transitions or edges, and vice versa.

If g_x and g_y, respectively, compute the differences in the x and y components for the depth image gradients of y and $\hat{y}$:

$$L_{grad}(y, \hat{y}) = \frac{1}{n} \sum_{p}^{n} | \, |g_x(y_p, \hat{y_p})| + |g_y(y_p, \hat{y_p})| \, |_1 \quad (3)$$

The structural similarity index(SSIM) loss, L_{SSIM}, measures the similarity between the predicted depth map and the groundtruth depth map in terms of luminance, contrast, and structure. The SSIM loss function encourages the network to learn a depth map that is visually similar to the groundtruth depth map. This loss is defined as:

$$L_{SSIM}(y, \hat{y}) = \frac{1 - SSIM(y, \hat{y})}{2} \tag{4}$$

C. Implementation Details

Weights Initialization: The weights of the decoder are randomly initialized at the start of the training process. This means that the values of the weights are not pre-determined, but instead are set randomly to provide a starting point for the optimization algorithm to improve upon.

ADAM Optimizer: The ADAM optimizer is used to optimize the weights of the decoder during training. The learning rate for the optimizer is set to 0.0001, which determines the step size taken during each update of the weights. The ADAM optimizer also uses two parameters, β_1 and β_2, which are set to 0.9 and 0.999, respectively. These parameters determine the decay rates for the moving averages of the gradient and its square.

Batch Size: The batch size for the training data is set to 8, which means that the optimizer updates the weights after processing a batch of 8 images. This helps to reduce the memory requirements for training, as the optimizer only needs to store the gradients for a small batch of images at a time.

Trainable Parameters: The model has a total of 42.6 million trainable parameters, which are optimized during training to improve the performance of the model. These parameters include the weights of the layers of the model, which are updated using the ADAM optimizer.

Number of Epochs: On the NYU Depth v2 dataset, a fine-tuning approach was used to train the encoder-decoder model for 5 epochs. The number of epochs determines how many times the model is trained on the entire dataset. Increasing the number of epochs can help improve the model's accuracy by allowing it to learn more from the data.

D. Evaluation Methods

The metrics used for evaluating the model are [2]:

The average relative error (rel): measures the average percentage difference between the predicted depth values and the ground truth values. Mathematically, it is defined as:

$$rel = \frac{1}{n} \sum_{p}^{n} \frac{|\hat{y_p} - y_p|}{y_p} \tag{5}$$

Root mean squared error (rms): measures the average magnitude of the errors between the predicted values and the actual values. The formula for rms is:

$$rms = \sqrt{\frac{1}{n} \sum_{p}^{n} (y_p - \hat{y}_p)^2} \tag{6}$$

The average $\log_{10}$ error: measures the accuracy of a prediction model by calculating the average logarithmic difference between the predicted and true values, and is useful for comparing models across a wide range of values.

$$average\ (log_{10})\ error = \frac{1}{n} \sum_{p}^{n} log_{10} \left(\frac{|y_p|}{|\hat{y}_p|} \right) \tag{7}$$

Threshold accuracy: compares the predicted label to the true label of each sample by computing the maximum ratio of the predicted label to the true label, or vice versa, and considers the prediction to be accurate if the maximum ratio is less than a predetermined threshold value. The threshold accuracy can be defined mathematically as such:

$$\text{Percentage of } y_p \quad \text{s.t.} \quad \max \left(\frac{y_p}{\hat{y}_p}, \frac{\hat{y}_p}{y_p} \right) = \delta < thr, \tag{8}$$

where:
y_p : the true label of the p-th sample
$\hat{y_p}$: the predicted label
δ : the threshold accuracy, which represents the percentage of samples that meet the criterion specified
thr : 1.25, 1.252 or 1.253

III. Dataset

A. Description

The original paper showcased two different datasets over which the authors have tried to train the depth estimation algorithm, in order to evaluate their performances.

The **KITTI** dataset [4] is a benchmark dataset designed for evaluating computer vision algorithms for autonomous driving tasks. It consists of a diverse set of real-world data, including stereo images, LiDAR point clouds, and GPS/IMU measurements, collected from a moving vehicle. The dataset provides ground truth annotations for a variety of tasks, including 3D object detection, 2D object detection, and road segmentation, making it a valuable resource for developing and evaluating computer vision algorithms for autonomous driving applications.

The **NYU Depth v2** dataset [5] is a large-scale indoor scene understanding dataset that provides RGB-D images and ground truth information for a variety of computer vision tasks, including semantic segmentation, depth estimation, and 3D reconstruction. The dataset consists of over 1,400 densely labeled indoor scenes, captured with a Microsoft Kinect sensor, covering a wide range of indoor environments such as homes, offices, and laboratories. The ground truth annotations include pixel-level semantic labels, surface normals, and depth maps, enabling the development and evaluation of advanced algorithms for indoor scene understanding.

The latter was used for this project, given the fact that the paper authors discovered that the NYU Depth v2 yielded better results for this task, due to its added depth map and larger size.

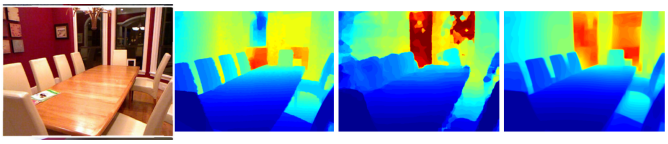

Fig. 4: Example image from the NYU V2 dataset with corresponding depth map [2]

B. Data Augmentation

Data augmentation is a common technique used to prevent overfitting and improve the generalization performance of neural networks. However, not all types of transformations are suitable for depth estimation tasks since distortions in the image domain do not always have meaningful geometric interpretations on the ground-truth depth.

Therefore, the authors selected a set of transformations that are appropriate for depth estimation, including horizontal flipping and color channel permutations. They excluded image rotation as it introduces invalid data for the corresponding ground-truth depth and set the probability of horizontal flipping to 0.5 and the probability of color channel augmentation to 0.25 [2].

IV. Results

For this section, we will be showcasing the results of evaluating the monocular depth estimation model using a selected set of random images from the NYU Depth V2 dataset. For our implementation, the terms 'DenseNet169' and 'DenseNet121' will be present, although the original paper is also based on the DenseNet169 model.

The evaluation process involves measuring various error metrics to assess the performance of the model. The key metrics considered are δ_1, δ_2, δ_3, rel, rms, and $\log_{10}$, which provide insights into the accuracy, precision, and overall quality of the depth estimation. Also, for each run, 9 random images from the test set will be selected, so for the qualitative evaluation the images for each network will differ.

Where:

- δ_1 : percentage of pixels with a ratio between ground truth and predicted depths less than 1.25
- δ_2 : percentage of pixels with a ratio between ground truth and predicted depths less than 1.25 squared
- δ_3 : percentage of pixels with a ratio between ground truth and predicted depths less than 1.25 cubed
- rel : mean absolute relative difference between ground truth and predicted depths
- $rmse$: root mean squared error between ground truth and predicted depths
- log_{10} : mean absolute logarithmic difference between ground truth and predicted depths

A. Qualitative Evaluation

For the depth estimation visualisation, the Jet colormap was used. The colormap assigns different colors to different depth values, providing a visual representation of the distance of objects from the camera:

- closer objects are represented by shades of blue, indicating their proximity to the camera
- intermediate objects are represented by shades of green and yellow, indicating a moderate distance from the camera
- farther objects are represented by shades of red, indicating their distance from the camera

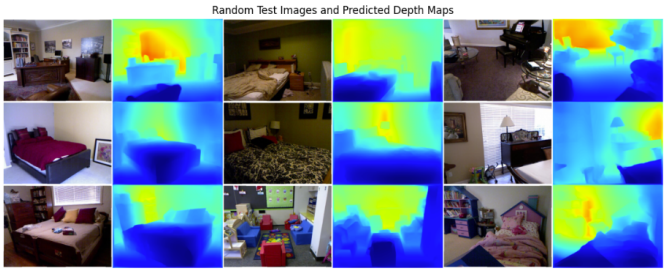

Fig. 5: Estimated depth maps for DenseNet169

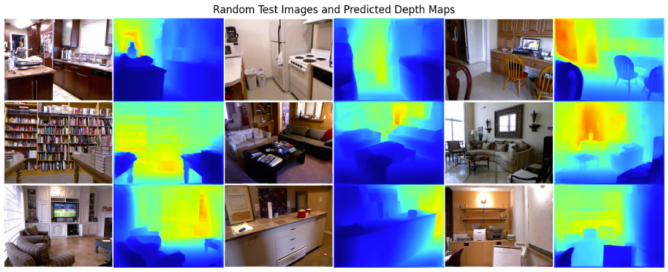

Fig. 6: Estimated depth maps for DenseNet121

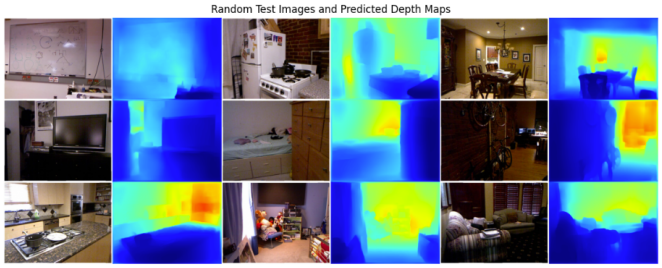

Fig. 7: Estimated depth maps from the NYU Depth v2 pretrained model [2]

In comparison to the original NYU model presented in the paper (Figure 7), both DenseNet169 (Figure 5) and DenseNet121 (Figure 6) models showcased similar performance in estimating depth information from RGB images. These models demonstrated satisfactory results overall, effectively capturing the relative distances between objects in the scenes and preserving fine details.

Although both models produced good results, it can be observed that the DenseNet121 model produced slightly sharper estimations, particularly in terms of object contours. This indicates that the DenseNet121 model has a better ability to capture fine details and boundary information in the depth estimation process.

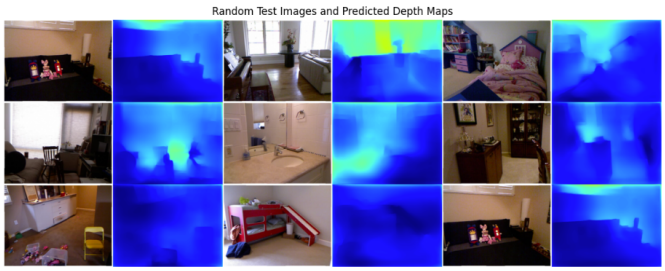

Fig. 8: Estimated depth maps from the KITTI pretrained model [2]

However, when evaluating the pretrained model on the KITTI dataset, the predicted depth maps displayed lower accuracy compared to the ground truth depth values - the contours aren't sharp and no farther objects are detected. The authors state that one possible reason for the limited success of their method in surpassing the state-of-the-art on this particular dataset is attributed to the inherent characteristics and limitations of the provided depth maps.

B. Quantitative Evaluation

Metric	DenseNet169	DenseNet121	Paper NYU	Paper KITTI
δ_1	0.8033	0.8108	0.8407	0.0037
δ_2	0.9596	0.9642	0.9721	0.0170
δ_3	0.9912	0.9921	0.9721	0.0448
rel	0.1412	0.1396	0.1259	0.7246
rms	0.5246	0.5023	0.4712	2.3185
log_{10}	0.0614	0.0597	0.0551	0.592

TABLE I: Evaluation Results

Both DenseNet169 and DenseNet121 models exhibit similar performance in capturing the relative distances between objects in the scenes effectively and preserving fine details. They achieve relatively high scores for δ_1, δ_2, and δ_3, indicating that a significant percentage of the predicted depths fall within the prescribed thresholds. These models demonstrate good accuracy in estimating depth values, as evidenced by the lower rel and rms scores. They also perform well in estimating logarithmic depth values, as indicated by the log_{10} scores.

In the context of our results, the average relative error provides insight into the overall accuracy. A lower average relative error indicates a better alignment between the predicted depth values and the ground truth values, implying higher accuracy in capturing the relative distances between objects in the scenes. Both DenseNet models exhibit relatively lower average relative errors compared to the model trained on the KITTI dataset.

A lower RMS error indicates that the predicted depth values closely align with the ground truth values, implying higher accuracy in capturing the depth information from RGB images. From these results, it can be observed that both DenseNet169 and DenseNet121 models exhibit relatively low RMS errors and the results are comparable with the ones from the original paper.

A lower log_{10} value indicates better performance, as it indicates a smaller discrepancy between the predicted and actual depth values on a logarithmic scale.

Our evaluation results indicate that both the DenseNet169 and DenseNet121 models in our implementation achieved similar performance to the models presented in the original paper. Specifically, the DenseNet121 network exhibited slightly better performance compared to DenseNet169. However, it is important to consider that the original authors had a much longer training duration, which could have contributed to their higher accuracy and convergence. Thus, it is crucial to take into account the training duration when interpreting and comparing the results between different implementations.

The quantitative results obtained from our evaluation reflect the lower quality images seen in Figure 8 with missing details from the KITTI pretrained model.

V. Conclusions

In conclusion, our implementation of depth estimation using the DenseNet169 and DenseNet121 models, fine-tuned on the NYU dataset, has demonstrated promising results. Despite the shorter training duration, our models have shown good performance in estimating depth information.

Both models have shown the capability to capture the relative distances between objects in the scenes accurately, while preserving fine details. The evaluation metrics, such as δ_1, δ_2, δ_3, average relative error (rel), root mean squared error (rms), and logarithmic error (log_{10}), indicate the models' accuracy in estimating depth values from single RGB images.

Our use of pre-trained models and transfer learning techniques has enabled us to achieve results similar to those reported in the original paper. This demonstrates the effectiveness of transfer learning in reducing the time and resources needed to develop accurate depth estimation models.

References

[1] Vulli, Adarsh, et al. *"Fine-tuned DenseNet-169 for breast cancer metastasis prediction using FastAI and 1-cycle policy."* Sensors 22.8 (2022): 2988.

[2] Alhashim, Ibraheem, and Peter Wonka. *"High quality monocular depth estimation via transfer learning."* arXiv preprint arXiv:1812.11941 (2018).

[3] El-Yabroudi, Mohammad Z., et al. *"Guided Depth Completion with Instance Segmentation Fusion in Autonomous Driving Applications."* Sensors 22.24 (2022): 9578.

[4] Geiger, Andreas, et al. *"Vision meets robotics: The kitti dataset."* The International Journal of Robotics Research 32.11 (2013): 1231-1237.

[5] Silberman, Nathan, et al. *"Indoor segmentation and support inference from rgbd images."* ECCV (5) 7576 (2012): 746-760.

[6] Huang, Gao, et al. *"Densely connected convolutional networks."* Proceedings of the IEEE conference on computer vision and pattern recognition. 2017.