Monocular Depth Estimation: Analyzing contribution of PackNet

Ruben Rodriguez Buchillon
Stanford University
coconutruben@cs.stanford.edu
https://github.com/coconutruben/cs231a

Abstract

Depth estimation is key for many autonomous tasks - from driving, over indoor navigation, to grasping and manipulation. Monocular depth estimation (depth estimation coming from a single view) is an interesting area that has seen many recent contributions for the potential upsides it provides - single camera systems are cheap, ubiquitous and require no calibration or correspondence matching. Contributions then center around improving the inherit issues of monocular estimation, from potentially low accuracy and coarse depth values, to the requirements of ground-truth data for supervised learning, and the inherit scale ambiguity present in single view reconstructions. In this project, I explore monocular depth estimation through the lense of 'PackNet' - a deep network architecture introduced by the Toyota Research Institute (TRI) in recent publications to learn a network to infer depth from monocular systems. In my work I adjust their architecture to me used on another dataset, and only use the Encoder/Decoder structure from their work to produce depth estimates. I show that at least with minimal learning, the architecture's results don't achieve the same results as a simpler baseline (with, and without pre-training). This is then concluded by suggesting next steps to explore why there is a performance delta, and whether it's really due to the structure, or an artifact of the environment, or restricted learning resources.

1. Introduction and Motivation

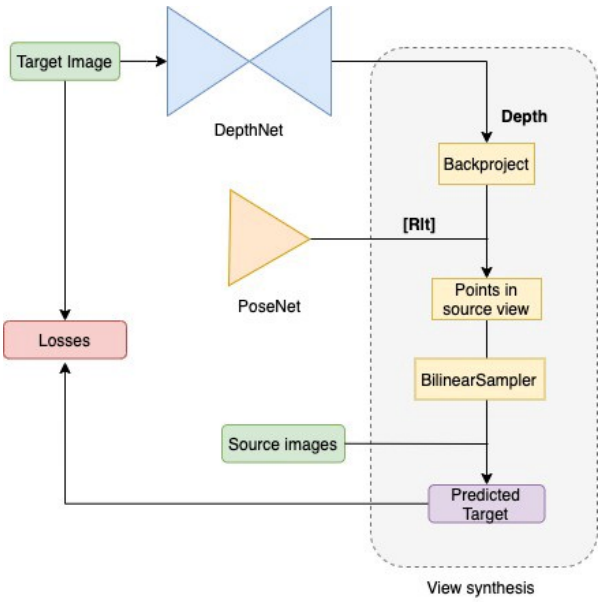
For this project, I want to replicate and analyze the contributions of the Toyota Research Institute (TRI) 2020 paper [4] on monocular depth estimation. The paper introduces a new network architecture (PackNet) to improve depth estimation on those images. I want to experiment with PackNet, see whether the performance matches the paper's claims, and whether the architecture is generalizable and produce similar results on other datasets (Make3D [10][11], NYU Depth v2 [7]). Motivated by the second part of the CS231A offering at Stanford (Winter 2021), I also want to see whether the learnt parameters generalize well in a transfer learning setting - by using the weights trained with their data to test on one of the previously mentioned datasets. I want to approach this in order to further my understanding of a deep neural network architecture based approach to depth estimation.

1.1. Problem Statement

This projects attempts to reproduce the results introduced in the TRI 2020 paper [4] on monocular depth estimation, and analyze the two main technical contributions of that paper - a new network architecture, and a new loss - and their impact on the 3D depth map reconstruction.
Understanding and analyzing the contribution of the network the paper introduces (PackNet) can be done not only on the dataset that the paper introduces, and others used in the paper ([14][2][3]), but also on unseen data as all its requires are depth images and ground truth labels. Through reproducing that architecture, and passing it through those works, I aim to understand this architecture in particular, and residual neural networks in general better, and collect qualitative data to see what the new networks contributions yields (in terms of test performance, and generalizability).
Analyzing the contribution of the loss is more constrained to the dataset the paper introduces and the KITTI [14] as it relies on instantaneous velocity data available between the images to address scale ambiguity.

2. Background and Related Work

Monocular depth estimation has been around for a while and gained many recent contributions to the area [15]. In general, these methods can all be grouped into the standard three categories: supervised, semi-supervised, and unsupervised. The semi-supervised, and unsupervised methods generally fall into a similar framework, where the supervision happens through reconstructing a target image. In the unsupervised case, the target image tends to come from a sequence of monocular images e.g. a video sequence. In the semi-supervised case, the supervision tends to come from the other other camera in a stereo setup.



(a) sfm pipeline

Figure 1: general structure for many approaches
source: tinyurl.com/coconutruben2-231a1

The problem itself is also inherently ambigious to scale, which requires algorithms and methods to come up with ways to address this. Common approaches include as hinted at before retrieving groud-truth data for supervision, using stereo rather than monocular vision during training, or using sequences with some ground truth annotation e.g. lidar. The requirements, though strictly only required during training and evaluation, do introduce extra complexity to setup and training, limiting one of the main advantages of monocular depth estimation: the simplicity of the setup.
Next, let's discuss residual neural networks, and why we want to use them here (not just 'PackNet', but many of the recent contributions to monocular depth estimation). The basic idea is that in a very deep network, gradients tend to become very small as they travel towards the intial layers. This can limit learning in two fundamental ways: 1. the weights don't change a lot as the gradients magnitude is vanishingly small 2. other layers might start to go overboard in their adjustments as the initial layers lag behind [12]. To combat this, one way is to use skip connections, or more formally a residual neural network. Here, we don't a layer in the network does not learn the mapping $f(x) = y$ where x is the input and y is the output, but rather $f(x) + x = y$ i.e. by passing x forward, we only need to learn the residual function. This has spawned a variety of very successful deep neural networks that also work very well for monocular depth estimation [15][5]

'PackNet' itself is an extension of the ideas introduced with ResNet [5]. Namely, it relies on a residual neural network architecture rather than a simple deep neural network. Unlike common residual networks, 'PackNet' does not use pooling and striding to reduce the dimensionality, but rather relies on their notion of 'packing'. This also allows for 'unpacking', rather than unsampling. The motivation there is that this avoids loss of information [4].

3. Contribution Analysis: PackNet

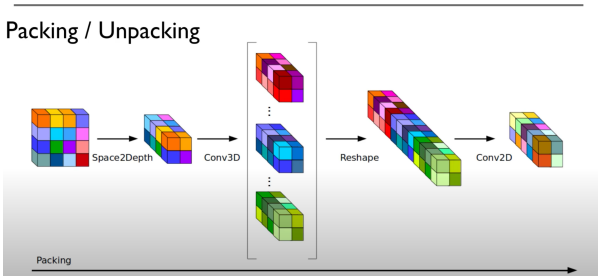
The paper has multiple contributions as they will outline: a new network structure, a new loss, and a new dataset (with lidar data and from multiple cities)[4]. For this project I'm mostly interested in exploring the network's contribution, though the velocity loss contribution also needs to be highlighted.

3.1. velocity loss

The idea is simple: rather than replying on expensive data to scale the estimate, scale the estimate by introducing a penalty on the pose network as the system goes from one frame to the next. Getting instantaneous velocity sensors and information is much cheaper and widely available than depth sensors. Similarly, if the scale is off, the true velocity information will disagree with the pose change from one frame to the next and thus induce the network to not only learn to retrieve depth estimates, but *metrically accurate* depth estimates.

3.2. network

The main contribution that the network highlights is its superior ability to act as an auto-encoder as it avoids the loss of information. In normal operations, when striding and pooling is happening, some pixels receive less attention than others. While the reduction in dimensionality is desired to detect a lower dimensional feature representation, the loss of information is less so. The authors here tackle this issue by folding and packing dimensions on a channel into new channels, rather than having large striding and pooling. Similarly, on the decoder side, the decode then unpacks that information back, rather than having to generate new information through upsampling, to get back to the high dimensional representation. This further allows for a more meaningful usage of 3d convolutions on the tensors. This is because there now is a clear relationship between the channels (as the information across multiple channels originally came from one channel before packing occured). With this technique, at least on the samples shown in the paper, the quality of the reconstruction of the auto-encoder is impressive. In the final section I talk about how validating and tuning this auto-encoder capability is key to determin-



(a) PackNet

Figure 2: Schematic of packing operations
source: youtube.com/watch?v=8_Vw6HqmhGY
(TRI presentation)

ing whether this structure has fundamental advantages over simpler networks.

4. Technical Approach

The goals of the project are to understand monocular depth estimation better, and to do so by diving deep into the TRI 2020 paper [4]. This required using a structure that lets me experiment with 'PackNet' - the main network contribution of the paper - and a baseline to see the performance, how the network works, and where things can be tuned for performance.
Ultimately, the proposed setup is to leverage prior work from the Stanford CS231A course [**?**] and reuse the homework notebook on monocular depth estimation to provide a framework to train and baseline on the NYU dataset [7]. This allowed me to train and test 'PackNet''s structure (the encoder/decoder layers) and compare them to the 'DenseDepth' [1] baseline that the class used for their assignment. It also provided a simple environment to experiment with the network.
The original idea was to first prove that 'PackNet' can outperform 'DenseDepth', and subsequently investigate what leads to these performance improvements. This was an ill-posed approach in that 'PackNet' does not outperform 'DenseDepth', neither in my setup with NYU, nor on online benchmarks with KITTI [14][8]. This meant that the same analysis could still be carried out to some extent, though the baseline was always beating the 'PackNet' result. It also allowed for more investigation into debugging why 'PackNet' was performing poorly.
The network used for 'PackNet' here is taken from the tri implementation [6] with a few modifications

- a version 'C' is introduced that sets all skips to 0 (turns the network into a plain deep network rather than a residual)
- the second to last layer is returned rather than the last, as our ground-truth depth maps come at half the resolution. This allows us to keep the guarantee that we do not upsample, but rather use the unpacking operations and stop when the right output resolution is found
- a small bug in the version 'B' is fixed that prevented the system from producing the right skip2 outputs

This overall leads to a network with 72 million parameters, and 2.5 hours of training time (compared with 'DenseDepth' clocking in at 1.3 hours of training time) per epoch.

4.1. Abandoned Approaches

This section outlines other approaches that for various reasons failed, but might be insightful in other ways to tackle the question of whether this structure provides tangible benefits.
synthetic temporal context
An approach was to make a synthetic temporal context window for NYU, or Make3D, so that the entire pipeline on the TRI paper [4] could be evaluated, rather than just the encoder/decoder layer, and not the velocity loss introduced.
This was cancelled because the main motivation for the velocity loss is to leverage it when ground truth data is absent, though it is available on these datasets, so initial evaluation needn't be blocked on that. However, the problem posed on the TRI paper about the lack of datasets with velocity annotation is a real one, so there is a motivation in generating synthetic frames and velocities to remove another source of uncertainity for evaluation during training: faulty instantaneous velocity information from sensors.
reproduction of full KITTI and DADD
This approach was to fully reproduce the results of the paper on the two main datasets used by the authors. This was cancelled because of the potentially low learning rewards (academic learning for this paper's author in this case). If the reproduction works - and the TRI team has done a fantastic job releasing code, configs, and docker containers online [6] - then I will have spent hours and computing resources on something that doesn't bring me closer to understanding their solution. If it doesn't work, it doesn't enable me to debug or understand the failure reasons more. Therefore, this was cancelled in favor of smaller experimentation as outlined above
parallel implementation, more datasets
This approach would have seen me expand their implementation, and their baseline to run on pytorch's parallel GPU system [9] to accelerate learning on multiple datasets. This was cancelled as 1. not enough computing resources were acquired, so at most one GPU could be used anyways, and 2. as expressed above, the baseline consistently overperformed in the experiments, motivating debugging and experiments on the basic experiments over expansion.

5. Experiments

This section outlines the base experiments and subsequent experiments that were run before diving into the results. Subsequently, you can find environment, test parameters, and runtime information towards the end of the section. It should be outlined that one of the main achievements that 'PackNet' claims is that it does not require pre-training the structure. Therefore, the baseline below with 'DenseDepth' uses a version without pre-training and learns all weights from scratch.

1. The base experiment was running 'PackNet' vs 'DenseDepth' in the framework provided through the class, and comparing the results. Subsequent experiments then deal with the details of 'PackNet' and their contributions

2. 'PackNet' provides a second version where rather than summing in the skip connections, they are padded on as another dimension. This experiment runs that second version and compares the output

3. 'PackNet' uses sigmoid as an activation function for the depth prediction (output) layers, which might suffer from diminishing gradients. This experiment is replacing that activation with a leaky ReLU

4. 'PackNet' predicts the inverse of the depth by default, rather than the depth directly. This experiment tackles replacing that with predicting the depth directly

5. 'PackNet' as a residual decoder uses skip connections. This experiment removes skip connections to see the plain network performance

5.1. Results

experiment 1: basic
The first experiment already showed that 'PackNet' does not outperform 'DenseDepth' in this setup. Even though 'PackNet' spent almost 2x more time learning for the epoch, the average loss was 0.612 (compared to 'DenseDepth' at 0.401). There are two peculiar things when observing the learning on 'PackNet'. 1. From the first few batches the loss per batch improves drastically, but then quickly settles in and only slowly increases over the remaining thousands of batches. This is in contrats to 'DenseDepth' that shows a clearer, steeper drive towards a reduced loss. 2. The variance between individual batches remains high throughout, something not seen on 'DenseDepth'. This means that while the average loss was already trending towards $0.7x$, 'PackNet' could often be seen still having $0.8x$ batches. Overall, the gradient steps appear much less guided.

(a) DenseDepth (b) PackNet

Figure 3: Qualitative Depth Map Analysis experiment 1

(a) DenseDepth (b) PackNet

Figure 4: Qualitative Depth Map Analysis experiment 4

experiment 2: version 'A' of 'PackNet'
This experiment produced comparable results to experiment 1 (the average loss was 0.623). It appears that skip connections being an extra dimension or being summed into the dimension don't change the performance too much.

experiment 3: leaky ReLU replaces sigmoid
This experiment was aborted after the initial batches. The initial average loss for the first half an hour hovered at around ≈ 0.66 and was showing the same patterns described above. While ultimately it might have helped a bit, the patterns were all the same and the changed activation did not seem to be the main culprit explaining the large performance delta.

experiment 4: non-inverted depths
The next experiment changed the output layers of the depth prediction from returning the inverse of a sigmoid activation to returning the actual sigmoid activation. This produced worse results, at an average loss of 0.731.

experiment 5: non-inverted depths
This experiment was also aborted after the first half an hour, because the average loss was significantly higher than before (average loss > 1.21) and showed no signs of drastically improving with subsequent batches. The residual structure thus seems to contribute a great deal in making sure that the large pipeline actually works as intended and can have meaningful performance even on the first batch.

5.2. Analysis

The main question that the experiments raise is why does 'PackNet' perform worse than 'DenseDepth', even though it has two main advantages:
1. structurally, it is a much larger network with many more parameters, and therefore should exhibit larger expressivity
2. the packing/unpacking layers avoid the loss of information in a way that the 'DenseDepth' does not
From the observations, it appears that the issue is that 'PackNet' on this setup gets into a rather diffuse (high variance) state and does not move out of it in a meaningful way batch after batch. Thus while there is some improvement over the entire epoch, the main improvements happens very early on, and the subsequent batches don't guide the network nearly as strongly. This would open up the question of whether the network is having very small gradients, or a disadvantageous learning rate so that any insights are lost when the next batch comes. Another possibility might be that the network simply requires many epochs to properly leverage its size and expressivity. While this might be the case, it would also raise questions on 'PackNets' complexity if extra complexity only leads to requiring extra training to achieve the same result as a simpler network. Another area that might cause issues might be the loss and what the network is trying to optimize against. On a preliminary experiment, I noticed that the predicted depths all hovered around one singular value - for the entire image. This might mean that the loss is inducing the network to learn the mean depth of the image, rather than the depth of specific sections. Lastly, some of the pre-processing might be interfering with the packing layers. 'PackNet' uses channels to create more feature channels and avoid information loss, so it might be an issue that we randomly swap around channels, if the relationship between those channels were beneficial for the network. On that last poin though, I would suspect that a swapped channel simply requires a different layer of neurons to be active, and therefore this preprocessing should not strongly interfere with the results.

5.3. Computing Environment

The notebook is adapted, and basic sanity checks were run on Google Colab Pro, to make sure processing and network structure work as intended.
The actual experiment is run on a Google Cloud Instance with 8 vCPUs, 52 GB memory, and one Nvidia V100 GPU, to leverage the cuda framework interface on pytorch.
This is then overseen through the notebook on a remote host, and debugging happens through ssh, and architecting the code in the notebook so that each cell reloads all contents that it needs, to allow for runtime reusage.

5.4. Tests and parameters

The tests conducted mirror the class assignment [**?**] in that they simply take a split from the NYU dataset, and learn from there the depth prediction, using the same loss. To keep the systems comparable ('PackNet' from this paper's experimentation and 'DenseDepth' from the assignment as a baseline), the system is run for 1 epoch, with a $lr = 0.0001$ and a batch size of 4.
The batch-size has interesting impacts on both the runtime and the memory constraints. A lower batch-size (2) does not reduce the runtime enough to justify it. A larger batch-size (8) expands the runtime on 'PackNet' at least to 8 hours (up from 2.5h), while not improving the loss incurred for the initial batches. Any batch-size larger than that would have motivated multiple GPUs and implementing the system as a parallel GPU architecture, as it runs out of memory on 'PackNet'.

6. Conclusion

In conclusion, the project has helped me a lot learn about the current state of the art in monocular depth estimation, the limits inherit with the approach, and practical experience with implementing and tuning a model for monocular depth estimation - albeit the results were not quantitatively very fruitful. It has also allowed me to get experience with the theoretical side of residual neural networks, and pratical experience implementing one and tuning it.
The experiment with skip connection removals shows that the underlying idea of leveraging residual neural networks to improve the models learning ability works in practice, and that the structures benefits do generalize, even if the overall performance of the network does not.

7. Takeaways and Next Steps

7.1. Technical Steps

The obvious next step is seeing if the networks size is the main contributing factor to the bad performance, and slow learning. This could be achieved by letting it actually run for 20-30 epochs (not even the whole 100 suggested in [4]).
On a more fundamental level, it appears that 'PackNet' relies heavily on the claim that it has a very small loss reconstruction loss when trained as an autoencoder. The next step there is validating that claim across multiple sets of datasets, including the ones suggested initially for expansion here [7][10][11]
I think the biggest takeaway for me is the need to produce clear environments, to progressive tune and evaluate models, and modifications, and their overall impact. The structure I chose to build upon here limited that to some

extent i.e. training takes a long time, and that cost is hard to reduce (financially, and timewise). However, even if those are addressed, there is still a need to develop a pipeline and tooling to let one quickly iterate over results and see the impact on smaller batches, that still have some significance. The authors at TRI [4] for example provide a sanity check dataset, a full dataset, and nothing in between. The obvious need here is for a small dataset and hardware environment that allows for rapid prototyping.

7.2. Beyond Monocular Estimation

Lastly, two other papers that I came across, one the paper motivating DenseDepth [1] and one motivating focusing on stereo for depth estimation [13], do raise some interesting questions about the architecture overall. DenseDepth achieves (both in the school assignment baseline I used here, and online [8] better performance than PackNet on the NYU dataset [7]. If this holds across multiple datasets, then the premise of leveraging deeper and harder to train (the 'PackNet' authors use 8 GPUs in parallel during training) networks might not yield the extra benefits that are desired - it might at least motivate having an extra layer that decides based on the complexity of the task to use a much simpler architecture. Nvidia researchers [13] on the other hand argue that even given the advances in monocular depth estimation, the importance of stereo cannot be understated, and that the inherit limitations on monocular estimation limit the accuracy that could be achievable. Given my results here I'm curious to dive into that topic and see the benefits and complexities of stereo next. Even the better results here on 'DenseDepth', or the claimed larger scale results of 'PackNet' when properly trained appear much more coarse than would be suitable for any but the most basic 'object avoidance' tasks (e.g. autonomous driving, as many of the research motivates), but would not be suited for anything finer grained.

References

[1] I. Alhashim and P. Wonka. High quality monocular depth estimation via transfer learning. *CoRR*, abs/1812.11941, 2018.

[2] H. Caesar, V. Bankiti, A. H. Lang, S. Vora, V. E. Liong, Q. Xu, A. Krishnan, Y. Pan, G. Baldan, and O. Beijbom. nuscenes: A multimodal dataset for autonomous driving. *CoRR*, abs/1903.11027, 2019.

[3] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele. The cityscapes dataset for semantic urban scene understanding. *CoRR*, abs/1604.01685, 2016.

[4] V. Guizilini, R. Ambrus, S. Pillai, A. Raventos, and A. Gaidon. 3d packing for self-supervised monocular depth estimation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[5] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[6] T. R. Institute. PackNet SFM. `https://github.com/TRI-ML/packnet-sfm#references`, 2020. [Online; accessed 6-February-2021].

[7] P. K. Nathan Silberman, Derek Hoiem and R. Fergus. Indoor segmentation and support inference from rgbd images. In *ECCV*, 2012.

[8] paperswithcode. KITTI benchmarks. `https://paperswithcode.com/sota/monocular-depth-estimation-on-kitti-eigen`, 2021. [Online; accessed 14-March-2021].

[9] PyTorch. MULTI-GPU EXAMPLES. `https://pytorch.org/tutorials/beginner/former_torchies/parallelism_tutorial.html`, 2017. [Online; accessed 3-March-2021].

[10] A. Saxena, S. Chung, and A. Ng. Learning depth from single monocular images. In Y. Weiss, B. Schölkopf, and J. Platt, editors, *Advances in Neural Information Processing Systems*, volume 18, pages 1161–1168. MIT Press, 2006.

[11] A. Saxena, M. Sun, and A. Y. Ng. Make3d: Learning 3d scene structure from a single still image. *IEEE Trans. Pattern Anal. Mach. Intell.*, 31(5):824–840, 2009.

[12] C. Shorten. Introduction to ResNets. `https://towardsdatascience.com/introduction-to-resnets-c0a830a288a4`, 2019. [Online; accessed 3-March-2021].

[13] N. Smolyanskiy, A. Kamenev, and S. Birchfield. On the importance of stereo for accurate depth estimation: An efficient semi-supervised deep neural network approach. *CoRR*, abs/1803.09719, 2018.

[14] J. Uhrig, N. Schneider, L. Schneider, U. Franke, T. Brox, and A. Geiger. Sparsity invariant cnns. *CoRR*, abs/1708.06500, 2017.

[15] C. Zhao, Q. Sun, C. Zhang, Y. Tang, and F. Qian. Monocular depth estimation based on deep learning: An overview. *Science China Technological Sciences*, 63(9):1612–1627, Jun 2020.