**ECE8443 Pattern Recognition Project Report**

**Student:** Alexander W Feild

* I use the royal “we” to refer to myself as a project team in this report

**MSU NetID:** awf97

**MSU ID:** 902290841

# **Introduction**

The purpose of this effort was to develop a method that can train a model to produce stereo disparity maps without ground truth disparity maps. The unsupervised metric used was image reconstruction. Additionally, reinforcement learning was used and the problem formulated as a non-linear optimization problem. Our goal is to evolve this method to non-aligned image pairs and optical flow. However, stereo pairs are a simpler case and we used Kitti 2015 for this project. This project report compares results to a state-of-the-art method that relies on labelled data, Hierarchical Neural Architecture Search for Deep Stereo Matching. We discuss the primary problem in reinforcement learning, predicting future reward, and limitation of our unsupervised reconstruction metric. We outline which steps and potential improvements we will experiment with next. Code for this project can be found here: <https://github.com/feildawproton/StereoRL>

# **Background**

* Bridging Stereo Matching and Optical Flow via Spatiotemporal Correspondence doesn’t use ground truth [reference]
* Stereo disparity vs optical flow (x, y) and non-aligned camers
* This method is also theoretically less sensitive to textures than SIFT (need reference and definition) and other deep learning classifiers. This is because it is composed of per-pixels agents and in theory disparity and flow estimations might naturally become smooth.
* Reconstruction similarity as a stand-in for disparity map

# **Methods**

* Ideas and architecture
* Software
  + Started with the keras tutorial on deep q learning
  + <https://keras.io/examples/rl/deep_q_network_breakout/>
  + Almost nothing about our codebase is the same
  + Also borrowed values from this tutorial with pytorch
    - <https://pytorch.org/tutorials/intermediate/reinforcement_q_learning.html>
    - Used a CNN for the same problem
    - Also Deep Q-Learning
  + Main idea from PixelRL [reference], but could not adapt the code because I was too unfamiliar with Chainer and the project structure so I recreated it in Tensorflow and Keras
* Reference the codebases that I started from
* This method doesn’t require ground truth
* Random actions are the same across all pixels. This was done to avoid scrambling the image and making the matching task harder on the FCN.
* Talk about numba here?
* Reward is the similarity between the left image and a recreation (from the left image and the map)

We use a decaying probability to determine if a random step is taken or if the agent takes an action (put in equation here) (adopted from). Decay is reset on every image. Therefore, random steps become a part of this agent’s reconstruction search. We tried decaying the probability of a random step over an entire epoch a got worse results (not reported here). This agent may rely on random steps because of its limited ability to predict future results. However, this needs to be explored further and fair consideration given to how random steps should be applied.

While we will expand this method to non-aligned cameras we will assume aligned stereo pairs for this project. Additionally we will only be producing disparity maps that are aligned on the left image. These disparity values are enforced to be negative (i.e. index to the left). The disparity map then recreates the left image using the right image as a source of pixel values (this is where the indices point). Each entry in the map contains a relative index. Relative indices must be added to each agent’s location to get global indices. If global indices are out of range of the right image, then nothing is drawn. A recreation of the left image, from the right image and using the map is what is used for calculation reward.

Since we use q-learning, the agents cannot output a map directly. Instead, they output the probability of a reward value for taking a particular action. In our case the particular action is a step that updates the disparity map. Therefore, the map is iteratively updated. This is a challenge to learning because the reward for moving in a particular direction is most likely non-linear. That means reward will go down before it goes up

# **Experiments**

* we used best case results from LEAStereo
* describe dataset
* train/test partition
* include here the tests and results of LEAStereo
* test both training on (online) and training off
  + because with unsupervised learning we can leave training on when an agent is deployed or update it periodically
  + \
* An epoch is all training images
* Include # of iterations per image
* Random action decay values.
* Kitti 2015 dataset

Experiments were ran on a single NVidia GTX 1080 with 8GB of memory. Limits on the number of filters per convolutional layer, filter size, batch size, and image size had to be observed in order to not run out of memory. Additionally, limits on the number of epochs per model in order for runs to be completed in time for updates. Learning rate for the Adam (reference paper) optimizer was also kept high, perhaps higher than optimum, for the same reason.

We also wanted to get Bridging Stereo Matching and Optical Flow via Spatiotemporal Correspondence [reference] working using the paper’s corresponding github repository. This required rewriting kernels to update to a newer version of CUDA than what the reference project uses. Additionally this required changing scripts and compiling C packages. However, dependencies were not resolved in time for results analysis.

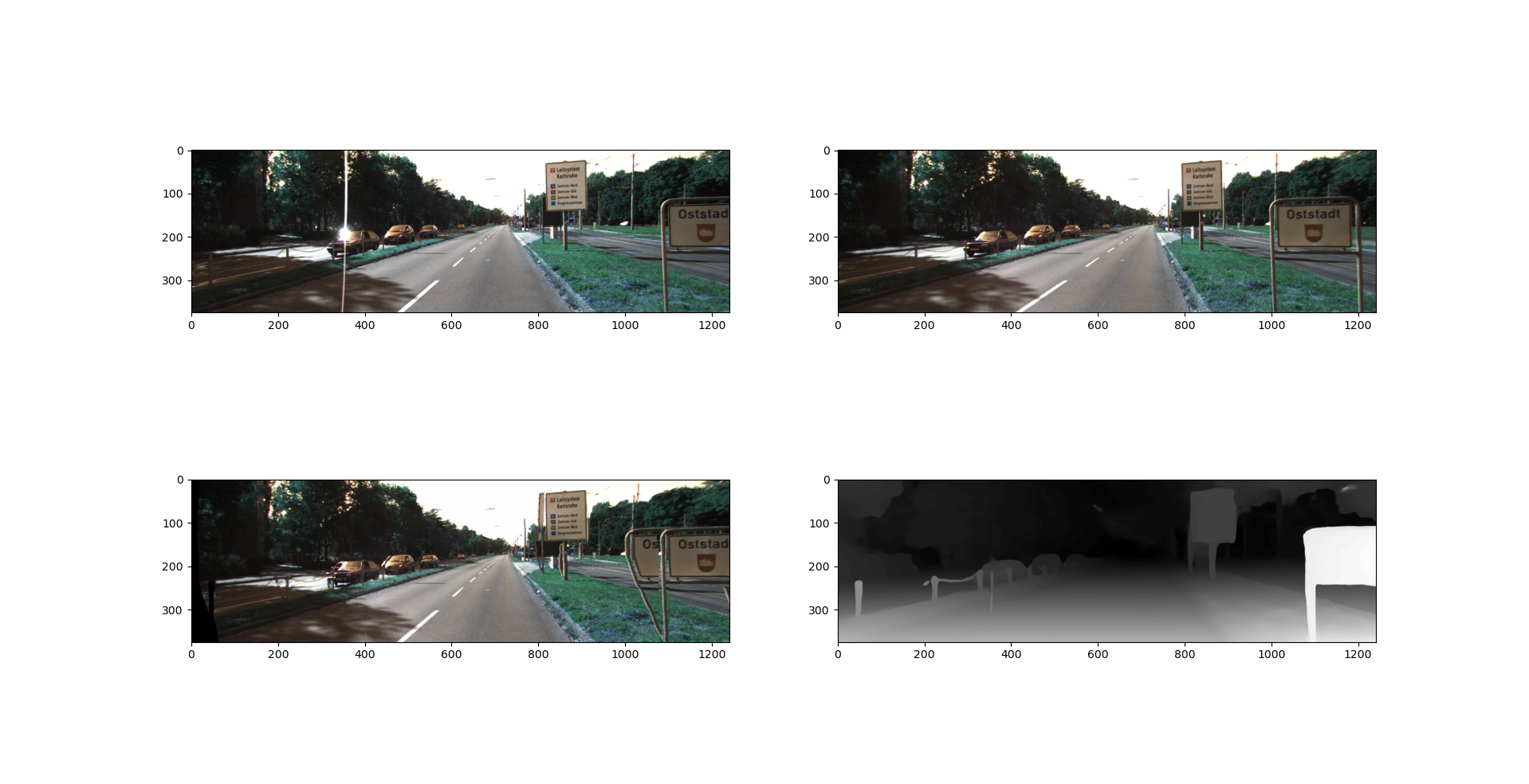
Tried getting the Efficient Deep Learning for Stereo Matching [reference] repository working. However, this would have required updating the whole project to the latest version of Tensorflow. We did not have update completed in time for analysis.

# **Results Analysis**

* talk about why we got the results we needed
* the image reconstruction metric that I used could probably be improved
  + reference paper here
* perhaps could have done better if we had more GPU memory and/or more GPUs
* When looking at LEAStereo we have to flip the values to be negative
  + And a lot of other stuff (put in the readme stuff here?)

## Reconstruction Similarity Function

We used the results from LEAStereo [reference] for comparison to our results. We also used LEAStereo to baseline the utility of our reconstruction similarity function. LEAStereo is the most performant of the methods tested on the Kitti 2015 dataset that also has its code readily available. It relies on ground truth data. LEAStereo outputs positive valued disparity maps that are aligned to the left image and point to the right image. This means that disparity values will need to be made negative. Additionally, disparity maps are stored as 16 bit pngs. Depending on the package used to open disparity images, the values may need to be scalled by 1/256. This is the case with Tensorflow’s image package but not with openCV. Indices are relative to their own pixel location; relative indices must be added to disparity pixel location to get the location on the right image.



Example result from LEAStereo on a test image pair. The top images are the left and right cameras. The lower right image is the disparity map (aligned with the left image and pointing to the right). The lower left image is the reconstruction of the left image using the map and the right image. Note the black on the left side of the reconstruction; this is expected. Also, not the error in duplicating the Otrstadt sign.

Assuming the results form LEAStereo to be good, we wanted to compare them against the reconstruction similarity between simply the left and right image. If the disparity estimation method works, and this one does, and our reconstruction similarity metric is valid, the similarity between the left image and the reconstruction must be higher than the similarity between the left and right images.

|  |  |  |  |
| --- | --- | --- | --- |
| LEAStereo Results | | full resolution | reduced res |
| left/right | Mean | 0.98879545 | 0.984352664 |
| Naïve | Stddev | 0.005035799 | 0.006459251 |
| Recreation | Mean | 0.964195463 | 0.968944835 |
| Naïve | Stddev | 0.01262049 | 0.011797016 |
| Recreation | Mean | 0.976425758 | 0.980948733 |
| Boost = 0.5 | Stddev | 0.008160826 | 0.007083238 |
| Recreation | Mean | 0.982540904 | 0.986950679 |
| Boost = 0.75 | Stddev | 0.00647343 | 0.005189422 |
| Recreation | Mean | 0.984008538 | 0.988391142 |
| Boost = 0.81 | Stddev | 0.006174637 | 0.00484361 |
| Recreation | Mean | 0.986209995 | 0.990551848 |
| Boost = 0.90 | Stddev | 0.005832285 | 0.004448236 |

In the table above we see the Left-Right similarity using a naïve metric was better than the reconstruction that was derived from the LEAStereo disparity map. This surely means our naïve metric is not a good one for disparity map and image reconstruction optimization. This is surely from pixel indices that go out of range. Parts of the left image will not be in the right image because they do no overlap. In the image reconstruction these pixels appear black. Our naïve metric is simply cosine similarity of color values. While values will be poorly matched between left and right images, it must be better (on average) than having a large number of black/blank pixels.

Because of this we tested boosting the reconstruction value of out of range pixels. Instead of zero (0) they returned the value “Boost” in the table above. In theory this number should be a balance between exploration (letting the model throw the correct pixels out of range), and preventing the model from gaming the reward function and throwing too many pixels out of range. This boost value should also result in better reconstruction similarity than left-right similarity. We went on to test a boost value of 0.9.

## Number of Epochs

We were not able to run a large number of epochs on our models. The most we were able to overnight was 5. The tables below compares 3 and 4 epochs and 4 and 5 epochs. We don’t see a significant difference. Perhaps this is to be expected. In the future we should run thousands of epochs with a lower learning rate.

|  |  |  |
| --- | --- | --- |
| Comparing Num Epochs (Naïve similarity, 4 inputs, 4 Normed Layers, 64 filters, 3x3 Kernel Dims, .001 LR) | | |
| Epcochs | Mean Similarity | StdDev Similaity |
| 3 | 0.985606322 | 0.006059277 |
| 4 | 0.98576417 | 0.006091498 |

|  |  |  |
| --- | --- | --- |
| Comparing Num Epochs (Naïve similarity, 4 inputs, 5 Normed Layers, 32 filters, 3x3 Kernel Dims, .001 LR, batch size 32) | | |
| Epcochs | Mean Similarity | StdDev Similaity |
| 4 | 0.985653708 | 0.006047823 |
| 5 | 0.985138951 | 0.006138407 |

## Number of layers

Our models had at most 5 hidden convolutional layer, along with batch normalization layers. This is not a lot for deep learning. This combined with our small kernels means that each pixel agent does not see a lot of the input images. Depths greater than 5 hidden layers were not attempted because of out-of-memory issues. We did not observe a difference depending on the number of hidden layers (again, perhaps because the difference was not significant).

|  |  |  |
| --- | --- | --- |
| Comparing Num Layers (Naïve similarity, 3 epochs, 4 inputs, 3x3 Kernel dims, 64 filters, .001 Learning Rate) | | |
| Normed Layers | Mean Similarity | StdDev Similaity |
| 3 | 0.985701586 | 0.006042564 |
| 4 | 0.985606322 | 0.006059277 |

## Batch Normalization

We wanted to see if batch normalization made a difference for our models. If batch normalization was used, batch normalization layers were inserted after every hidden convolutional layer. The table below shows a noticeable, though not significant, difference between using batch normalization and not.

|  |  |  |
| --- | --- | --- |
| Batch Normalization vs not (Naïve similarity, 3 epochs, 4 inputs, 3 layers, 3x3 kernel, 64 filters, .001 Learning Rate) | | |
| Batch Normed | Mean Similarity | StdDev Similaity |
| No | 0.984587062 | 0.006383839 |
| Yes | 0.985701586 | 0.006042564 |

## Number of Filters

For simplicities sake, and borrowed from PixelRL, every hidden layer used the same number of filters. The table bellow compares 32 and 64 filters per batch normalized layer. A significant difference was not observed. At depths greater than 4 layers, we could not use more than 32 filters or we would run out of memory on our GTX 1080.

|  |  |  |
| --- | --- | --- |
| Comparing Num Filters (4 epochs, 4 inputs, 4 Normed Layers, 3x3 Kernel, .001 Learning Rate) | | |
| Num Filters | Mean Similarity | StdDev Similaity |
| 32 | 0.985622946 | 0.005981981 |
| 64 | 0.98576417 | 0.006091498 |

## Similarity Metrics

As noted above our naïve similarity metric punished out of bounds pixels. However, this is inevitable. We tested not accounting for out of bounds pixels, boosting their reconstruction similarity to 0.9, and setting them to white in the reconstruction. The goal is to balance algorithm and model exploration with exploitation. We either did not achieve this balance with these metrics or there was some other problem that made the similarity metric irrelevant.

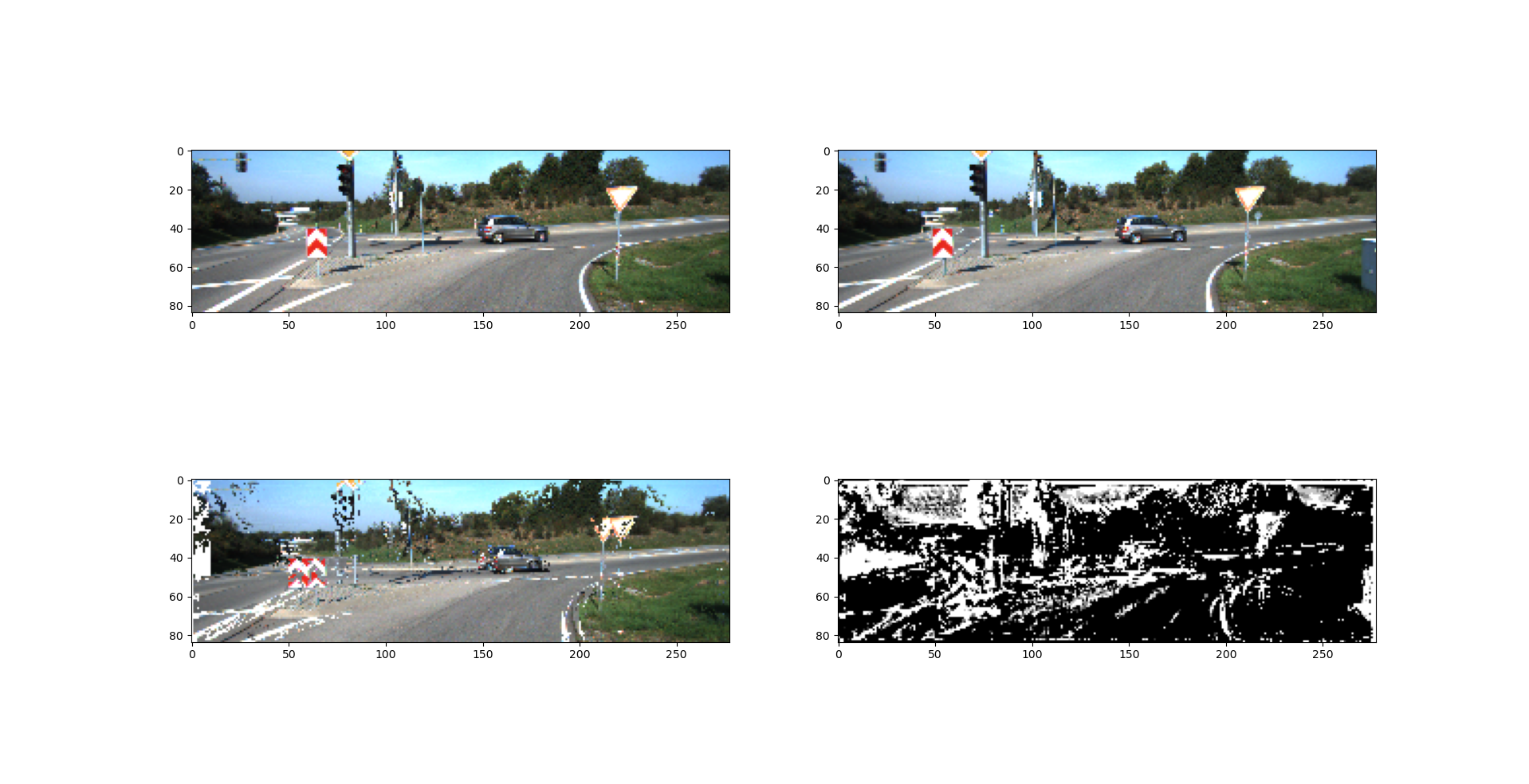
|  |  |  |
| --- | --- | --- |
| Compairing Similarity Methods (5 epochs, 4 inputs, 5 normed layers, 3x3 kernels, 32 filters, batch size 32, .001 Learning Rate) | | |
| Similarity Method | Mean Similarity | StdDev Similaity |
| Naïve | 0.985138951 | 0.006138407 |
| 0.9 Boost | 0.985942968 | 0.006033917 |
| White Background | 0.985977651 | 0.005776801 |

## Comparison to Nothing and the State-of-the-Art

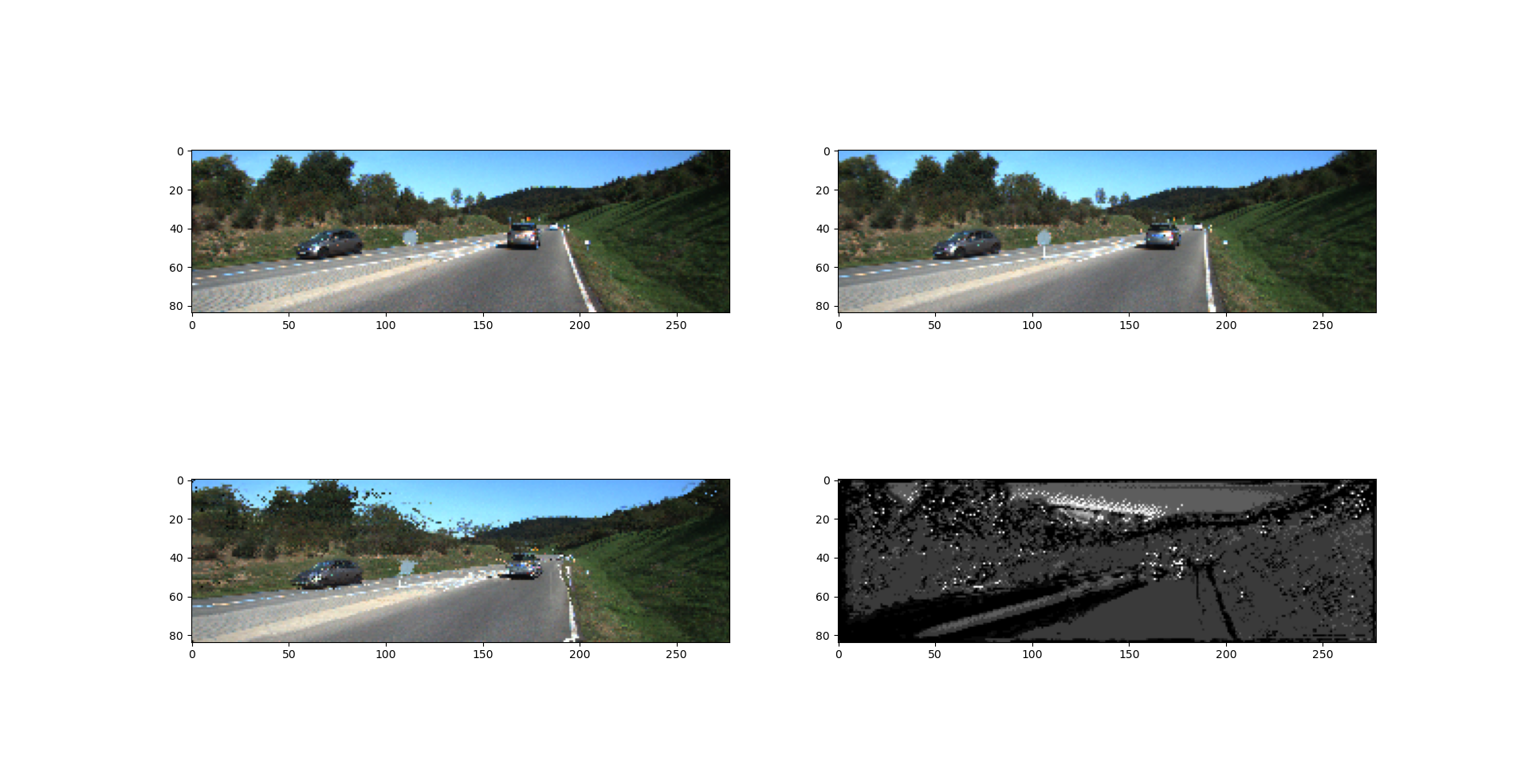
In the table below we report the similarity metric for our method, LEAStereo, and the Left-right similarity. Note these are for reconstruction, not for disparity estimation. Our method is noticeably better than doing nothing, though not significantly so. LEAStereo, a best in class Kitti 2015 disparity map estimator, did the best. The mean similarities of our method and doing nothing are outside the first standard deviation of LEAStereo’s reconstruction similarities.

|  |  |  |
| --- | --- | --- |
| Method | Mean Similarity | StdDev Similarity |
| Left-Right similarity | 0.984352664 | 0.006459251 |
| Our method 0.9 Boost | 0.985942968 | 0.006033917 |
| LEAStereo 0.9 Boost | 0.990551848 | 0.004448236 |

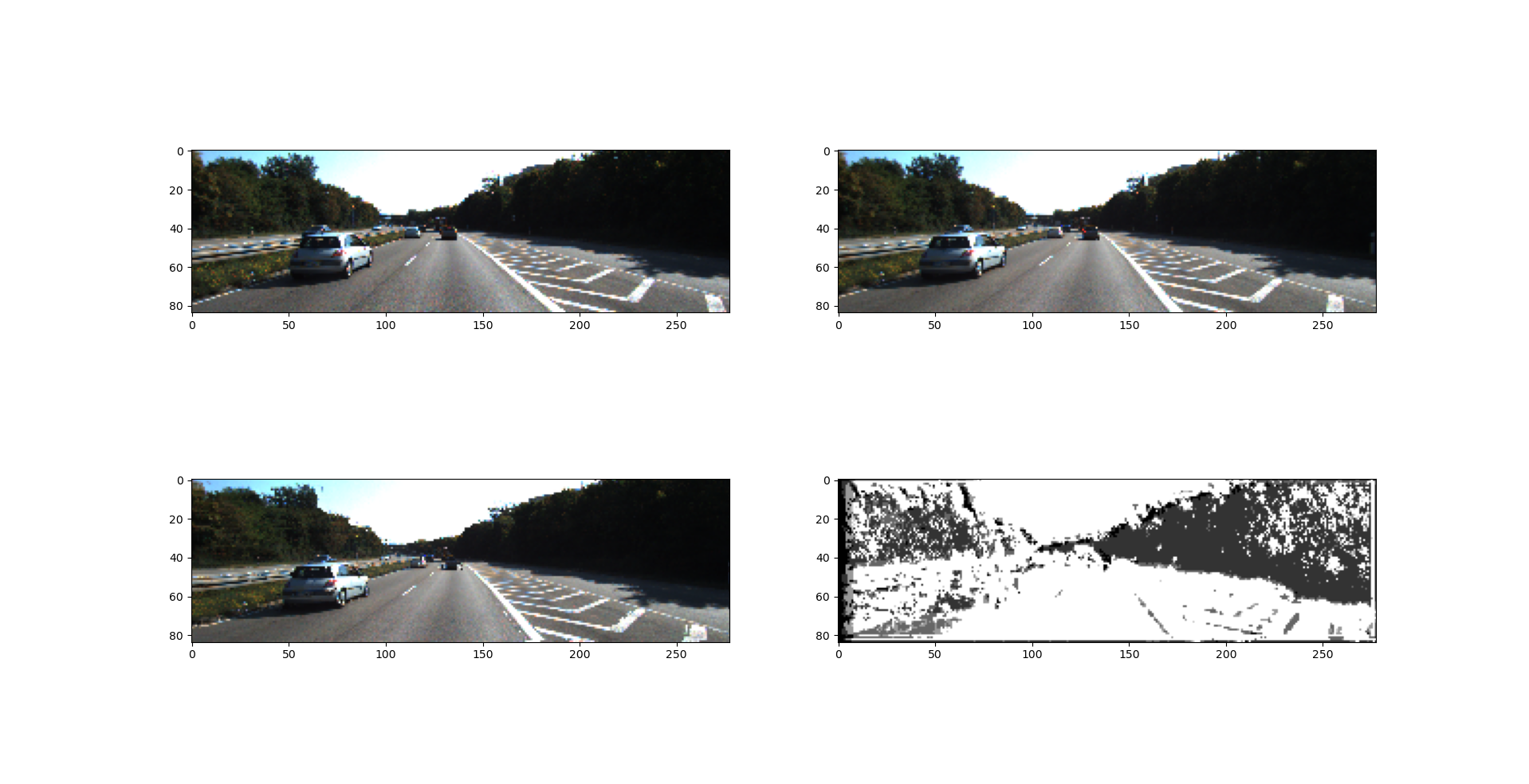
Subjective Analysis of Some of Our Reconstructions



This is an example test result from a 5 normed layer model with a similarity metric that substituted white pixels for out of range indices. Notice in the reconstruction that about half of the each street sign and some of the road marks are quite well aligned, while the others left in their original location. Perhaps this model would have done better with more iterations over these images or with more training epochs. We believe this clearly demonstrates learning the objective and that we are on a decent path.



This is an example from testing a 5 normalized hidden layer model, trained with 0.9 boost to out-of-range-indices for 5 epochs. Though the image is scramble a bit we can see that the agent did a decent job of aligning most of the image. Perhaps this model would have benefited from more training or more parameters.



This is an example test result from a model trained with 4 normalized layers with 64 filters each for 4 epochs. This agent did a decent job of moving the car over for alignment.

# **Conclusions**

* more epochs
  + better inclusion of future reward
* lower learning rate
  + combined with more epochs and we may get a better search
* more depth
  + especially for kernel size 3x3 (borrowed from PixelRL)
    - this means that each agent did not see a lot of the images
  + On a better compute platform perhaps (HPC?)
* Recurrent NN
  + In the form of ConvLSTM probably
  + Likely lead to better prediction of future reward
* Better thought out reconstruction metric
  + But don’t want to encourage the model to throw indices out of range because that is easier and gets sufficient reward
* Basically need a better estimate of future reward
* Need to test how random steps should be implemented
  + Compare over image, over epoch, over entire training
* This methods does not relying on camera alignment and should expand to optical flow (x, y indexing) fairly easily, though this would require more training and computation. This method also does not relying on knowing camera properties or even camera calibration. Cameras could even have separate properties, making this method useful in situations that more traditional methods would not be. This method would be a first good step in correspondence matching
* This method is also theoretically less sensitive to textures. Moving forward we could test this.
* Should we test disparity and flow smoothness. Is this a natural consequence of sharing the weights between all pixel agents?
* We could also compare to a classifier that outputs complete maps each iteration. We will still use the same unsupervised recreation metric.

# **References**

Xuelian Cheng, Yiran Zhong, Mehrtrash Harandi, Yuchao Dai, Xiaojun Chang, Tom Drummond, Hongdong Li, Zongyuan Ge, “Hierachical Neural Architecture Search for Deep Stereo Matching” *34th Conference on Neural Information Processing Systems*, 2020, Vancouver, Canada