Pitch and Roll Camera Orientation From a Single 2D Image Using Convolutional Neural Networks

Greg Olmschenk*, Hao Tang†, Zhigang Zhu‡

*‡The Graduate Center of the City University of New York
†Borough of Manhattan Community College - CUNY
‡The City College of New York - CUNY
*golmschenk@gradcenter.cuny.edu, †htang@bmcc.cuny.edu, ‡zhu@cs.ccny.cuny.edu

***Abstract*—In this paper, we propose using convolutional neural networks (CNNs) to automatically determine the pitch and roll of a camera using a single, scene agnostic, 2D image. We compared a linear regressor, a two-layer neural network, and two CNNs. We show the CNNs produce high levels of accuracy in estimating the ground truth orientations which can be used in various computer vision tasks where calculating the camera orientation is necessary or useful. By utilizing accelerometer data in an existing image dataset, we were able to provide the large camera orientation ground truth dataset needed to train such a network with approximately correct values. The trained network is then fine-tuned to smaller datasets with exact camera orientation labels. Additionally, the network is fine-tuned to a dataset with different intrinsic camera parameters to demonstrate the transferability of the network.**

I. Introduction

For many tasks in computer vision, the knowledge of the orientation of a camera can be an invaluable resource. In particular, the pitch and roll of a camera can provide much detail on the perspective of the image being viewed. One example would be a hand-held camera (e.g. a mobile phone camera) being carried by a visually impaired user. The angles of the camera would be useful for providing orientation information for object detection and navigation to the user.

Another example would be a camera overlooking a scene. One can imagine surveillance footage or news camera footage looking over a crowded gathering. Knowing the perspective helps to determine the size of objects in the scene and therefore detect, count, or otherwise label information.

Unfortunately, this information is not typically automatically known. In many cases an accelerometer may be onboard to determine the orientation, but many cameras have no such device, movement of the device can throw the orientation estimation off (such as while a robot or person is in motion), and images obtained secondhand (such as from a Google search) commonly do not have this meta data available. These values can be manually determined by annotating an image, but doing so requires extensive human effort and such annotations are prone to errors. The method given here works to provide a reliable measure of the orientation without any information beyond the image itself.

We have trained convolutional neural networks (CNNs) to recognize patterns in an image and automatically determine the roll and pitch of the camera from this information. The networks were trained on an extensive dataset of more than 400,000 images. The average predictive accuracy of the CNNs is within a few degrees of the true value. However, the roll and pitch in the data is not uniformly distributed. Specifically, due to the method of data collection, most of the roll and pitch values cluster around the same values, so simply knowing the average absolute error has little meaning because a different distribution would produce a different error. Thus, we use other metrics which we discuss in more detail in Section V-A. The initial networks were trained using an approximate dataset from an onboard accelerometer. This was used to train a network from scratch which was then fine-tuned to preform well on exactly annotated data. This network was also fine-tuned to a camera with different intrinsic parameters to test the ability of it being transfered to other cameras then the initially trained camera.

The main points that make this work unique are the following:

- Provides a mostly scene agnostic orientation detection method. (No explicit requirements are set on the image, though the vast majority of the examples are indoor images)
- Does not require knowledge of intrinsic camera parameters.
- Introduces a method to provide the large datasets needed for training a deep network for orientation, where others have discussed the lack of such datasets [2].
- Allows for real-time estimation through CNN inference.
- Proposes a novel use of CNNs in determining camera parameters.
- Tests the transferability of orientation estimation networks to different cameras.

The rest of the paper is organized as follows. First, we discuss what is new in our approach compared to the state-

of-the-art in Section II. We discuss the methodology in Section III. Section IV describes the data in detail and how it was prepared. In Section V we describe the specifics of each network trained and tested. Detailed results are given in Section V-A. In Section VI we discuss the meaning and value of the results. Finally, we conclude in Section VII.

II. Related Work

Fischer, Dosovitskiy, and Brox [3] proposed a convolutional neural network with the goal of estimating 2D image rotations from a single image. Their work consisted of artificially rotating images from image classification databases and building networks which could classify and regress to the rotation. Our work goes beyond this previous research in two main aspects. Firstly, our network works to take into account multiple orientations (both pitch and roll) rather than a single orientation change. Secondly, the rotations in the work by Fischer, Dosovitskiy, and Brox are 2D transformations. Due to the nature of their datasets, they did not have an option to apply 3D transformations. Our dataset provides 3D rotations that create perspective distortions and come from images taken in actual rotated camera poses. This provides a greater challenge which our network was able to solve. Our work also introduces a method to provide the large datasets needed for training a deep network for orientation. From this, the model can be fine-tuned on small manually annotated datasets.

Kendall, Grimes, and Cipolla [5] determine full 6-DOF camera localization using a CNN, however, their approach requires the network is trained on the same scene it is being tested on. Specifically, their network matches the position of the camera relative to a known database of images containing the same structure as is in the new test data. Our network is designed to determine orientation with no previous knowledge of the scene and without any database to match against.

Borji [2] used a CNN to detect a vanishing point. This work could possibly be a precursor to estimating orientation, but in their work they were only detecting a single vanishing point within the image frame, and only had success with images on roads. They discussed the need for more data to provide a more general approach, which is a problem we address in our work.

Other non-CNN methods to determine the orientation of the camera require extra information or specific scene types. For example, localization by structure from motion [14] requires multiple images at the very least where our method requires only a single image. Most methods which detect the world coordinate system vanishing points (from which the orientation can be calculated) require a scene with many conspicuous parallel lines running to these vanishing points [13] or make other assumptions about the content of the image. Our method makes no explicit assumptions about the scene.

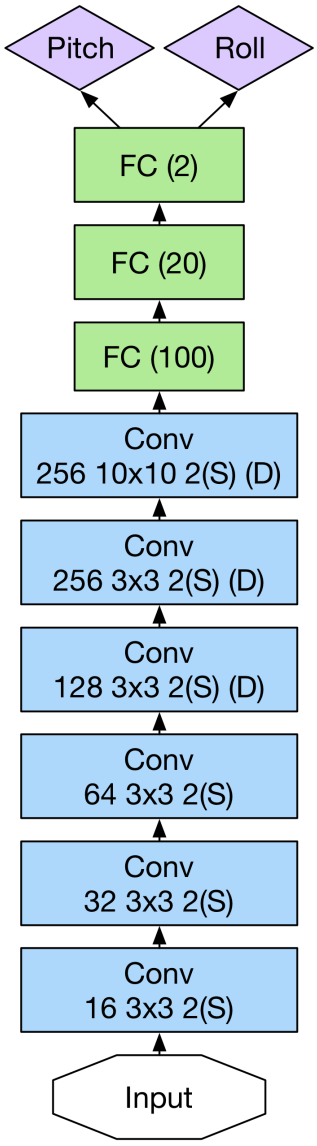

Figure 1: A diagram of our first CNN. "NxN" displays the kernel size of the layer. This value is preceded by the number of filters being output. Each convolution in this network is applied with a stride of 2. Layers with dropout during training display a "(D)".

III. Methodology

Our first convolutional network consisted of 9 layers. The network diagram can be seen in Figure 1. The structure design based on information in literature as well as our experimental trials. The first 6 layers were 3x3 convolutional layers of depth (number of filters) 16, 32, 64, 128 256, and 256. Each convolution was applied to the full depth of the previous layer. All 6 convolutional layers were applied with a stride of 2 to the layer before them. This is followed by three fully connected layers. These had a unit count of 100,

20, and 2, with the output of the last being pitch and roll. These consist of sets of weights applied to the activation of previous layers, in the form of $\mathbf{y} = W\mathbf{x} + b$. After each layer (excluding the output layer) a Leaky Rectified Linear Unit (ReLU) [8] activation function was applied. A ReLU was chosen due to it's effectiveness and simplicity [8]. The leaky variant produces an activation of

$$y_i = \begin{cases} x_i, & \text{if } x_i \geq 0 \\ \dfrac{x_i}{a}, & \text{if } x_i < 0 \end{cases} \tag{1}$$

with x_i being the input from channel i, a is a fixed parameter, and y_i being the output activation for channel i. The leaky activation combined with the momentum of the optimizer prevented neurons from remaining in a saturated (negative) state. Additionally, a 50% dropout [12] was applied to the final 3 convolutional layers. The dropout both prevents overfitting and improves the robustness of the network by reducing the networks reliance on a select few neurons. The dropout was only applied during training, while during test time all neurons were used (with weights properly scaled to account for total number of neurons being used). An Adam optimizer [6] was used for the parameter update due to its experimental effectiveness in bringing a network to convergence.

The second CNN involved a more complex structure, as seen in Figure 2. This network was similar to the first as there are again six main modules followed by a fully connected layer. In the first network, each module was a single convolution layer. In this network, each module consisted of four parts: a column convolution layer, a row convolution layer, an average pool layer, and a maximum pool layer. Both pool layers output a value for each kernel position of each input depth, with the value corresponding to their respective names in regards to their input. The average pool and max pool layers were followed by a 1x1 convolution[1] in all modules. The output of these four parts are then concatenated. The final two concatenations also have dropout applied during training.

It should be noted that receptive field of the convolutions for both networks is the same. That is, in both spatial height and width, although divided between row and column convolutions individually, they still contain 5 kernels of size 3 and 1 kernel of size 10. Breaking the convolutions in to both row and column components in the second CNN (the branching structure) requires fewer weights while maintaining the receptive field. This allows either faster training or more layers to be added.

The average pool and maximum pool can be seen as retaining a compressed version of previous layers for the new layers to use as input. There is no average pool in the

[1]This convolution allows for a dimensionality reduction an acts as a miniature fully connected layer from an aisle to the next layer's aisle.

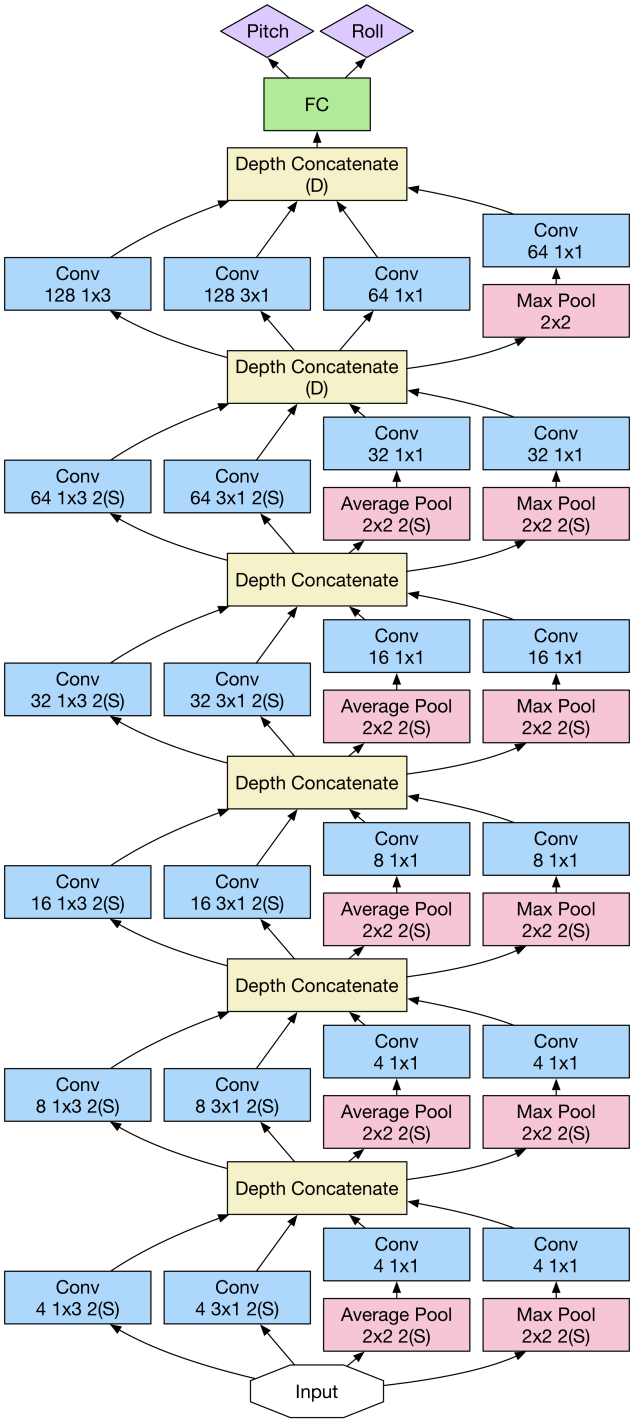

Figure 2: A diagram of our second CNN. "NxN" displays the kernel size of the layer. For convolutional layers, this value is preceded by the number of filters being output. Layers applied with a stride of two show a "2(S)". Those without this are a stride of one. Layers with dropout during training display a "(D)".

final module, as the striding is one, so it would only act to blur the existing information.

The leaky ReLU activations, optimizer, and dropout for this network were the same as for the previous network.

For all of the networks[2], mini-batching was used [7]. The mini-batching provides approximate global descent steps and the ability to utilize parallelization for processing speed while still allowing more rapid gradient decent by not needing to process the entire dataset.

IV. Data

As mentioned by Borji [2], one difficulty that arises in training a deep network to detect the orientation of the camera (or a similar task) is the lack of sufficient training data. To overcome this difficulty, we propose taking advantage of other sensors in existing large datasets to infer the information needed for the dataset to act as training data for our task.

The data used in this paper to train the initial network from scratch is the entirety of the NYU Depth Dataset V1 [9] and V2 [10]. This combined database consists of ~450,000 images taken from a Kinect device. Typical RGB data from this dataset is shown in Figure 3. The Kinect is most well known for its depth sensor, however, there is also an onboard accelerometer. We used this sensor to calculate the acceleration due to gravity so that the orientation of the camera could be estimated.

For the exact manually annotated data, we used the SUNRGBD V1 dataset [11]. This dataset includes data from various other datasets as well, including portions of the NYU Depth V2 [10], Berkeley B3DO [4], and SUN3D [15]. For this data, the transformation from the camera to the global reference frame has been manually annotated, so the roll and pitch of the camera can be easily obtained. Specifically, we use the small portion of the NYU Depth V2 which has been manually annotated to fine-tune the model, and test it on the Berkeley B3DO data. Both these use the Kinect V1 as is used with the accelerometer generated dataset. We additionally fine-tune the network to the Xtion device data in the SUNRGBD dataset. This allows us to test our model when applied to a different camera then was used for training the original network. Of the SUNRGBD dataset images, we used ~1200 for training and ~200 for testing from the NYU Depth dataset, ~450 for training and 56 for testing from the Berkeley B3DO dataset, and ~750 for training and ~250 for testing from the Xtion dataset.

For all datasets, an additional validation dataset was prepared of approximately equivalent size to the test data. These validation datasets were used for hyperparameter tuning.

A problem that arises when using the acceleration of the Kinect is that the acceleration comes from both gravity and the movement of the person who carried it during the data taking process. However, the acceleration from movement is minor in comparison with gravity in this case. And more importantly, we only used this large accelerometer dataset to train the initial network, which is then fine-tuned on the exact data from the SUNRGBD V1 datasets. When generating our large approximate orientation dataset, acceleration was assumed to be due only to gravity. Comparing the 1449 images that make up the portion of the NYU Depth V2 dataset which was manually annotated in the SUNRGBD dataset against our annotations from the accelerometer, we can check the error in the accelerometer generated orientations. Various measures of error in the accelerometer data from the true orientations can been seen in Table I: Mean error, mean absolute error (MAE), and root-mean-square error (RMSE). It seems that the error in roll is larger than in pitch. The distribution of the approximate dataset generated by the accelerometer data can be seen in Figure 4.

Measure	Combined	Pitch	Roll
Mean Error	N/A	+0.230	+1.771
MAE	2.226	1.543	2.910
RMSE	3.153	2.137	3.759

Table I: Error of accelerometer data from true orientation when comparing 1449 images.

During our experiments with the approximate accelerometer dataset, ~400,000 images were used for training and ~45,000 images were used for testing.

The variation in the data causes some difficulties. While the types of environments are extensive, the orientation of the camera tends to be neutral. That is, the Kinect tended to be angled near level with the ground and without much roll (zero pitch and zero roll). A different dataset would produce a different absolute accuracy using the same network. To account for this, we are not interested in the absolute error from the true value, but instead use baselines. In particular, we use the mean value over the entire set as the lowest level baseline, while the linear and two-layer networks provide additional measures of comparison.

V. Experiments

For the initial training and testing using the accelerometer generated dataset, five approaches were tested and compared. Our goal was to see how well our deep convolutional neural networks compared against other methods.

The first of the methods compared against was the baseline which simply uses mean values for pitch and roll in the test dataset. Such a choice of baseline is necessary due to the bias in the data a simple absolute angle accuracy means very little (as explained in Section IV). Additionally, it provides a way to see which networks have any impact at all. If a network cannot do better than this baseline, then the network can't be seen as having learned anything useful.

[2]Including the networks used for comparison which are detailed in Section V.

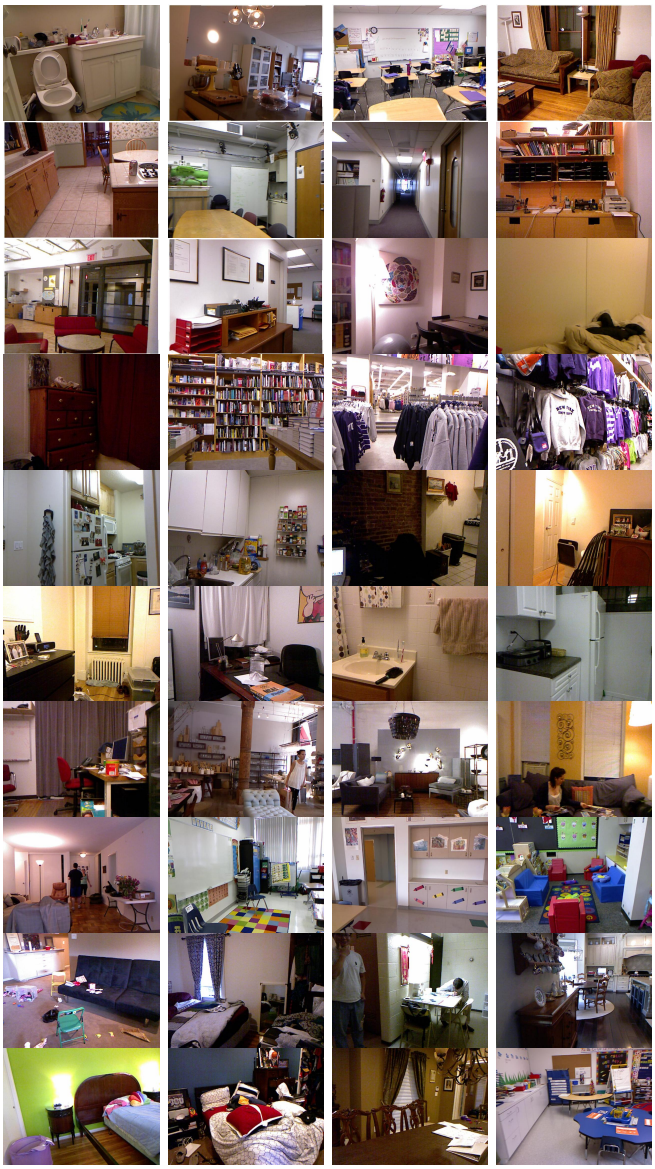

Figure 3: Typical example images from the database.

The linear regressor was of the simple form of $\mathbf{y} = W\mathbf{x} + b$. Here $\mathbf{y}$ contains the 2 values being sought, namely pitch and roll. $\mathbf{x}$ is the pixel values of the image under test in vector form. W contains the weights being trained and b is 2 biases, which are also trained. An Adam optimizer [6] was used for the parameter update.

The fully connected two-layer neural network started with the input being each color channel of each pixel in the image. This input is fully connected to 64 neurons. These neurons are fully connected to the 2 output neurons, one which outputs roll and the other pitch. The layer of the 64 neurons are passed through a Leaky ReLU. More formally, we have

$$\mathbf{y} = W_2 f_{\text{lReLU}}(W_1\mathbf{x} + b_1) + b_2. \quad (2)$$

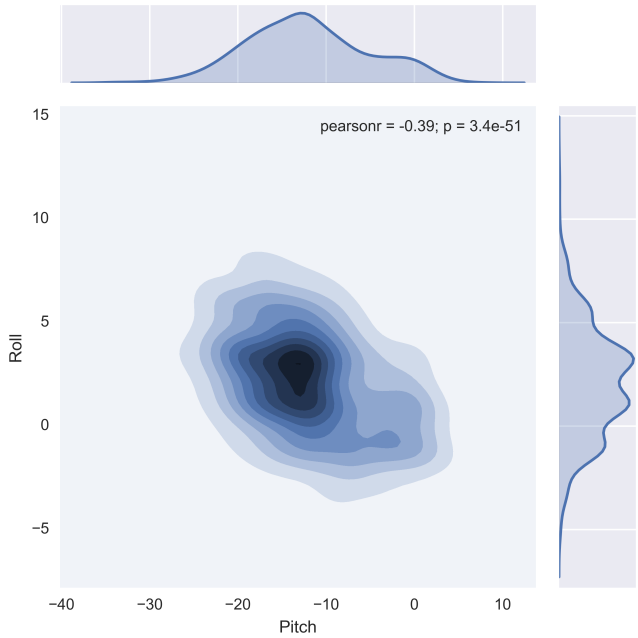

Figure 4: The distribution of the test data. Note that there are examples outside the contour, but relatively few in comparison with the number within the contour.

	RMSE		
Network	Combined	Pitch	Roll
Test Mean Value	8.618	7.117	2.887
Linear	12.868	13.681	12.000
Two-Layer	5.007	6.413	3.002
Branching Deep	3.988	5.069	2.471
Straight Deep	2.324	2.852	1.634

Table II: Results showing the root mean squared error of the predicted label from the true label in the accelerometer generated dataset from the complete NYU Depth V1 & V2 datasets.

An Adam optimizer [6] was used for the parameter update.

The details of the proposed two CNNs can be found in Section III: in these results we refer to the first network as the Straight Deep CNN and the second Branching Deep CNN.

For all of the networks, mini-batching was used [7]. All networks were constructed using the TensorFlow framework [1].

A. *Results*

Table II shows the results of our experiments comparing the various networks on the dataset generated using the accelerometer. Table III shows the results of the experiments when fine-tuning the network on smaller databases with exact orientations. In all experiments, due to the tendency of the data being centered around the neutral orientation,

Datasets				RMSE		
Test	Train	Finetune	Network	Combined	Pitch	Roll
SUNRGBD NYUD V2 14%	N/A	N/A	Test Data Mean Value	5.137	5.302	4.972
SUNRGBD NYUD V2 14%	Accel NYUD V1 & V2	SUNRGBD NYUD V2 86%	Straight Deep	3.613	3.090	4.069
SUNRGBD NYUD V2 14%	Accel NYUD V1 & V2	None	Straight Deep	4.161	4.045	4.274
SUNRGBD B3DO 11%	N/A	N/A	Test Data Mean Value	9.776	13.240	6.312
SUNRGBD B3DO 11%	Accel NYUD V1 & V2	SUNRGBD B3DO 89%	Straight Deep	7.649	10.198	3.609
SUNRGBD B3DO 11%	Accel NYUD V1 & V2	None	Straight Deep	7.498	10.001	3.522
SUNRGBD B3DO	N/A	N/A	Test Data Mean Value	11.339	12.489	4.767
SUNRGBD B3DO	SUNRGBD NYUD V2	N/A	Linear	60.976	62.098	59.833
SUNRGBD B3DO	Accel NYUD V1 & V2	SUNRGBD NYUD V2	Straight Deep	6.903	9.132	3.450
SUNRGBD B3DO	Accel NYUD V1 & V2	None	Straight Deep	6.379	8.372	3.360
SUNRGBD Xtion 33%	N/A	N/A	Test Data Mean Value	11.339	12.489	4.767
SUNRGBD Xtion 33%	Accel NYUD V1 & V2	SUNRGBD Xtion 66%	Straight Deep	8.402	11.203	3.958
SUNRGBD Xtion 33%	Accel NYUD V1 & V2	None	Straight Deep	9.034	12.234	3.683

Table III: RMSE for various combinations of training, testing, and fine-tuning datasets for various networks.

we optimized our networks to result in the lowest root mean squared error (RMSE). Thus the three values were measured for each network: pitch, roll, and combined values. The "combined" value is the error when taking into account both pitch and roll differences between the labels, and is the value that was optimized by our network. It should be noted that for all of these results, the networks were trained to determine roll and pitch simultaneously. There was not a separate network for each[3].

During the experiments using the accelerometer generated data, the deep CNNs had the highest accuracy for pitch, roll, and combined values. The straight deep neural network out performed the branching deep network by a significant amount. Possible explanations are discussed in Section VI.

It should be noted that the linear and two-layer networks failed to out perform the mean value in regards to roll. These networks are not guaranteed to find a set of weights can do even as well as this static value. Considering what a darker image overall and brighter image overall using the same weights in these networks would output should give some sense as to why this is.

In the fine-tuning experiments (Table III) we used the network trained on the accelerometer data and retrained the last few layers of the network on the annotated data. For each set of data, we provide the baseline value and the error in using the the accelerometer trained network with and without fine-tuning. The last two fully connected layers in the deep network were being fine-tuned in these trails.

During fine-tuning trials with the manually annotated data, the deep neural network had significant improvements over the baseline, but far less than the improvements seen on the network trained from scratch. Using an additional validation set, it seemed that underfitting was the largest cause of this discrepancy and perhaps the problem could be mitigated by fine-tuning larger portions of the network, though it seems larger numbers of samples are needed to effectively do this. This is discussed further in Section VI.

These networks also have the advantage of being able to run in real-time. The network is able to process an image in ~40ms on a consumer level laptop[4]. These were the speeds achieved without any special optimization for deployment purposes. Optimization was only performed for the goal of speeding training.

VI. Discussion

We see from our results in Section V-A that deeper networks tend to have better results. This makes sense as the deep convolutional network is able to detect features that would suggest orientation, such as vanishing point lines, where as the simpler networks cannot.

While examining these results, the distribution of the data should be kept in mind. Most of the pitch and roll values lie within a fairly narrow range of values. This both means that the accuracy has an absolute advantage, in that even simply selecting the mean value results in not large absolute errors, but it also presents a challenge to the network as any rotations that are recognized are very small rotations. Presumably, a deep network could more easily detect a large rotation than a small one, though explicit evidence for this is not given here.

In contrast, the deep CNNs could most easily determine roll, but have a more difficult time with pitch (though still doing far better than the other networks). This also is reasonable if we consider something like the recognition of lines. Most of the images were in a mostly neutral position. Due to this, any roll rotates lines parallel to the ground to a similar degree in the image, even if those lines are not horizontal in the image. A change in pitch results in the more complex behavior of moving vanishing points (i.e. changing

[3]Though, in the case of the linear classifier, outputting to two values is equivalent to having two separate networks.

[4]2 GHz Intel Core i7, 8 GB 1600 MHz DDR3.

perspective distortions) without as obvious of a change in individual lines in the image. If correct, it would then be reasonable to assume that had the network been trained on a database with a more uniform spread of rotations that the pitch and roll accuracies would have been more similar.

There was a interesting difference between the two deep CNN structures. Both did approximately equally as well in regards to estimating pitch. However, the straight CNN structure did significantly better than the branching CNN structure. Having estimated pitch more accuracy while doing worse with respect to roll, in this regard at least, makes the branching structure similar to the non-deep networks. One possible reasoning for this is that the 1xN and Nx1 filters do not take into account more complex spatial features as well as an area convolution despite acting on the results both row and column filters from the previous layers. However, the possible causes were not explored further in this work.

In the trials testing on the annotated data, the most notable feature is that fine-tuning resulted in no significant change in error. This seems to be caused by underfitting due to the small portion of the network being retrained. However, during extensive tests of fine-tuning, different numbers of layers being retrained and various sizes of layers were tested. Either no change in the training and validation were affected, or the network overfit the data. This suggests that the number of samples in these datasets were not sufficient to fine-tune the network effectively. This is reflected in the fact that the largest of these datasets (SUNRGBD NYU, $\sim$1200 training images) had the best fine-tuning results and the smallest (SUNRGBD B3DO, $\sim$450 training images) had the worst results. Additionally, the network without fine-tuning had a significant improvement over the baseline, which both shows that the accelerometer trained network is useful when applied to other scenes, but also that the fine-tuning provided little improvement due to the underfitting.

In the end, the most clear result of the experiments is the high level of accuracy of the CNNs when compared against the other networks in the accelerometer trials. Even though the data clusters around a neutral pose, the use of the CNNs to determine the minor changes in orientation provides a significant advantage.

VII. Conclusion

In this work, we proposed a method to automatically determine camera pitch and roll from a single 2D image without any database to compare to or explicit prior information about the image being considered. We have shown that deep CNNs produces significant improvements over various other approaches. Additionally, we have revealed interesting information in how different complexity levels of networks are able to determine information. We have devised a way to produce large datasets for training such orientation estimating networks. Additionally, this work shows that CNNs can be used to determine information about the camera itself, in addition to the scene it is viewing. We show the accuracy of these networks when applied to manually annotated datasets, both with the same camera as the main training set, and when used for a different camera.

VIII. Acknowledgments

This research was performed under appointments (to Tang and Hao) to the U.S. Department of Homeland Security (DHS) Science & Technology (S&T) Directorate Office of University Programs Summer Research Team Program for Minority Serving Institutions, administered by the Oak Ridge Institute for Science and Education (ORISE) through an interagency agreement between the U.S. Department of Energy (DOE) and DHS. ORISE is managed by ORAU under DOE contract number DE-AC05-06OR23100 and DE-SC0014664. All opinions expressed in this paper are the author's and do not necessarily reflect the policies and views of DHS, DOE, or ORAU/ORISE. Computing power for this research was partially supported by Azure for Research. This work is also partially supported by the U.S. National Science Foundation (NSF) through Award EFRI-1137172.

References

[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] A. Borji. Vanishing point detection with convolutional neural networks. *arXiv preprint arXiv:1609.00967*, 2016.

[3] P. Fischer, A. Dosovitskiy, and T. Brox. Image orientation estimation with convolutional networks. In *DAGM*, 2015.

[4] A. Janoch, S. Karayev, Y. Jia, J. T. Barron, M. Fritz, K. Saenko, and T. Darrell. A category-level 3d object dataset: Putting the kinect to work. In *Consumer Depth Cameras for Computer Vision*, pages 141–165. Springer, 2013.

[5] A. Kendall, M. Grimes, and R. Cipolla. Convolutional networks for real-time 6-dof camera relocalization. *CoRR*, abs/1505.07427, 2015.

[6] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

[7] M. Li, T. Zhang, Y. Chen, and A. J. Smola. Efficient mini-batch training for stochastic optimization. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 661–670. ACM, 2014.

[8] A. L. Maas, A. Y. Hannun, and A. Y. Ng. Rectifier nonlinearities improve neural network acoustic models. *Proc. ICML*, 30(1), 2013.

[9] N. Silberman and R. Fergus. Indoor scene segmentation using a structured light sensor. In *Computer Vision Workshops (ICCV Workshops), 2011 IEEE International Conference on*, pages 601–608. IEEE, 2011.

[10] N. Silberman, D. Hoiem, P. Kohli, and R. Fergus. Indoor segmentation and support inference from rgbd images. In *Proceedings of the 12th European Conference on Computer Vision - Volume Part V*, ECCV'12, pages 746–760, Berlin, Heidelberg, 2012. Springer-Verlag.

[11] S. Song, S. P. Lichtenberg, and J. Xiao. Sun rgb-d: A rgb-d scene understanding benchmark suite. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 567–576, 2015.

[12] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.*, 15(1):1929–1958, Jan. 2014.

[13] J.-P. Tardif. Non-iterative approach for fast and accurate vanishing point detection. In *2009 IEEE 12th International Conference on Computer Vision*, pages 1250–1257. IEEE, 2009.

[14] M. Tomono. 3-d localization and mapping using a single camera based on structure-from-motion with automatic baseline selection. In *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, pages 3342–3347, April 2005.

[15] J. Xiao, A. Owens, and A. Torralba. Sun3d: A database of big spaces reconstructed using sfm and object labels. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1625–1632, 2013.