CS 231a Project Final Report
6D Pose Estimation using PVNet

Abhinav Raghunandan Kannan
abhi74k@stanford.edu

Nicholas Moy
nmoy@stanford.edu

Abstract

In this project we implement PVNet, a neural network based solution for 6D pose estimation. The paper proposes a novel representation which robustly handles occlusion and truncation. Instead of directly regressing the position of the 2D key points on the image, PVNet predicts unit vectors from each pixel to each of the key points. Such a dense representation is found to be more robust. The paper uses a 2 stage approach: the 1st stage uses PVNet to produce an image segmentation map and unit vector prediction. The 2nd stage uses RANSAC voting mechanism to find key points on the image and the PnP algorithm uses the identified key points and 3D model to estimate the position and orientation of the object. In this project, we have re-implemented the PVNet architecture from ground up in PyTorch and used transfer learning for the initial weights from ResNet, RANSAC for keypoint identification and used OpenCV's implementation of PnP algorithm. With limited training we are able to achieve passable results in a few instances, although model performance falls short of the original paper.

1. Introduction

The problem of 6D pose estimation is estimating the position and orientation of the object in the camera reference frame. 6D pose estimation has many applications in the field of robotics. Previously proposed pose estimation methods are not robust against occlusion and truncation. In this project we implement PVNet[6], a neural network based solution to estimate the pose in the presence of occlusion and truncation. The paper proposes a novel vector representation of 2D keypoints. Learning the vector representation instead of directly regressing the keypoint locations will force the network to focus on local features of the objects and spatial relationship between the object parts.As a result, the location of an invisible part can be inferred from the visible parts and even represent object key points outside the input image.

For this project, our goal was to to explore the components of a state-of-the-art approach to the problem, and what factors might impact the performance of the resulting model. We implemented a a streamlined version of the PVNet pipeline described in the paper, replicating the core elements necessary to produce end-to-end predictions. We then sought to experiment with different training strategies and parameters to understand how well an unrefined prediction pipeline performs versus the multiple stages of refinement needed to achieve the results of original paper.

Our implementation achieves passable predictions, but performance is ultimately substantially worse than that achieved by paper authors, and prior approaches, with training time being the most important limiting factor.

2. Related Work

One of the pioneering papers in the field of 6D pose estimation is PoseCNN [9]. It introduces a CNN architecture to directly regress a 6D pose from a single RGB image. It localizes the object in 2D and predicts depth to the 3D location. For estimating 3D rotation which is non-linear, it discretizes the rotation space and casts the rotation as a classification task. The approach produces a coarse result and is also not robust against occlusion and truncation. Instead of directly regression 6D pose, keypoint methods adopt a 2 stage pipeline: they first predict 2D keypoints and then compute the final pose using the 2D-3D correspondances. Keypoint methods typically work well for objects with rich textures. PVNet falls under the category of keypoint methods. For textureless objects, recent works used semantic keypoints and CNN as keypoint detectors [7].

There are also dense approaches where every pixel makes a prediction to the desired output in a Hough voting scheme [3][1] uses a random first to establish 2D-3D keypoints using geometric constraints for such dense correspondences.

3. Approach

The key innovation of PVNet is to not try to predict the positions of the keypoint directly, but rather to predict a tensor of per-pixel Unit Vectors that point to the 2D projection of the keypoint in the image. Keypoints are then estimated in a second stage, with PnP pose estimation as the final step.

The workflow we implemented for the PVNet pipeline is illustrated in Figure 1. All of the code for training running the model can be found at https://github.com/abhi74k/pvnet/.

3.1. PVNet CNN

The first step of the PVNet pose estimation pipeline is a Convolutional Neural Network (CNN) that produces two outputs.
Class Segmentation: a per-pixel, per class probability mask.
Keypoint Vectors: a per-pixel, per-class, per keypoint 2D vector pointing in the direction of one keypoint for one class.

3.1.1 Model Structure

PVnet is based on the ResNet-18[5] model. The first stages of PVNet are identical to ResNet. First, RGB input goes through basic stride-2 7x7 Convolution, Batchnorm and Rectified Linear Unit (ReLU). Then it is iterated through 3 "Residual" layers of the same basic format:

- Stride-2 3x3 Convolution and 2x Channel Depth
- BatchNorm
- Rectified Linear Unit (ReLU)
- Stride-1 3x3 Convolution
- Batchnorm
- Residual "Add-Back"
- Relu

The residual add-back avoids the degradation problem, where especially deep neural networks paradoxically learn more slowly and less accurately than shallow models [5]. Unlike the original Resnet-18, the fourth layer is a 3x3 convolution with zero stride. To maintain the receptive field, the 3x3 kernel is dilated so that each pixel "covers" the same area of the original image as a 2-stride convolution.

Next, using this basic encoding from ResNet, instead of a fully connected convolution and averagepool, we perform two 3x3 convolutions, then reverse the downsampling through 4 "upsampling blocks" to return to the same dimensions as the input image. The Upsampling block is essentially to reverse the "residual block" convolutions, using 2x bilinear sampling at each layer for 4 layers. Skip connections are implemented whereby outputs from Residual Layers are added directly to output of the upsampling convolutions of corresponding size, right before it is upsampled.

The final output of [64,H,W] goes through two final 1x1 Convolution layers, one for Class Segmentation output, and one for Keypoint Vector output. Class segmentation output outputs one channel per class, where values represent probability that that pixel belongs to a given class. Final segmentation is given by the index $[C+1, H, W]$ where each channel represents class probabilities for one class. Keypoint vector output is of shape $[C \times V \times 2, H, W]$. That is, each pixel has vectors pointing to V keypoints for C classes, and each vector has x and y component to specify direction. The structure of the model is outlined in Table 2

3.1.2 Model Training

For Class Segmentation loss, we used Cross Entropy loss across all classes in the training list, plus a "null" class for background pixels. One important discovery in training the model was that a large proportion of the image was background pixels rather than any class of interest. As a result, when training with naive Cross-Entropy loss, the model quickly converged toward undesirable outcome of classifying every pixel as background. Modifying the Cross Entropy loss function to ignore background pixels altogether produced the opposite effect of classifying background pixels as objects. We achieved substantially improved performance by down-weighting, but not ignoring, the loss contribution from background pixels. We selected a relative weight of 0.2 for null pixels, based on the average mix of object vs background pixels across our training images.

As in the original paper, for Keypoint Vector loss, we used smooth L1 loss for every pixel, for every keypoint k.

$$\ell_k(w) = \sum_{k=1}^{K} \sum_{p \epsilon O} \ell_1(\Delta v_k(p;w)|_x) + \ell_1(\Delta v_k(p;w)|_y)$$

$$\Delta v_k(p;w) = \tilde{v}_k(p;w) - v_k(p)$$

Where $k \epsilon K$ refers to each of the keypoints for a given object of a given class, $p \epsilon O$ is each pixel in a given object class, w represents the parameters of PVNet, $\tilde{v}_k$ is the predicted vector, v_k is the ground truth vector, and $\Delta v_k(p;w)|_x)$ and $\Delta v_k(p;w)|_y)$represent the two elements of Δv_k, respectively.

Many methods use eight corners of a 3D bounding box as target keypoints. However, the original PVNet paper

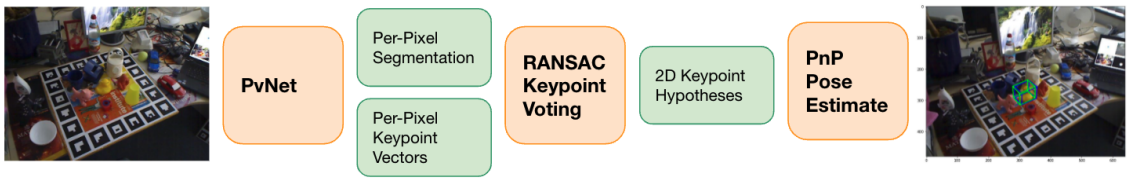

Figure 1. Overview of PVNet Pipeline

Layer	Output Shape
Image Input	[3,H,W]
Initial Convolution	[64,H/2,W/2]
Residual Layer 1	[64,H/4,W/4]
Res Layer 2	[128,H/8,W/8]
Res Layer 3	[256,H/16,W/16]
Res Layer 4	[512,H/16,W/16]
Final Convolution Layers	[256,H/16,W/16]
Upsample Layer 1	[128,H/8,W/8]
Upsample Layer 2	[64,H/4,W/4]
Upsample Layer 3	[64,H/2,W/2]
Upsample Layer 4	[64,H,W]
Segmentation Output	[C+1,H,W]
Vector Output	[C$\times V \times 2, H, W$]

Figure 2. PvNet CNN Structure

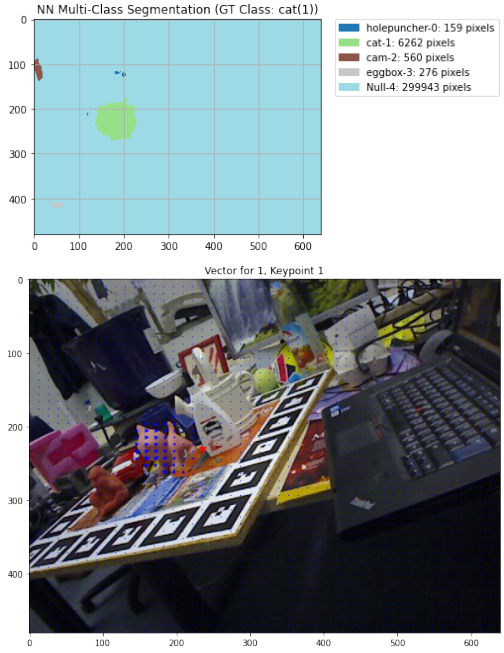

Figure 3. Example model outputs

[6] explores a new method of selecting keypoints on the surface of the object itself, which they observe led to significantly better less variance in predictions than training on corner keypoints. In our implementation, we ultimately only trained models on corner keypoints.

We initially computed this loss using predicted segmentation to determine the pixel set $p \epsilon O$, since this would ultimately be how the model output is used to predict keypoints. However, this compounded any loss errors from segmentation, and significantly increased the time it took to converge on a solution. For our final implementation, we used the ground truth class mask label directly so that this loss only captures vector errors.

We make one important simplification in our implementation that bears mentioning: we compute keypoint losses for all classes, but summed over the set of pixels O_{target} belonging to only one class – the class for which we have label data for that image. As described further in Section 4.1, we only have labels for one target class per input image in our training data. All other classes have zero values for class and vector ground truth, even though objects of those classes might be present in the image. For non-target classes, the effect is to ignore vector predictions outside of O_{target}, and inside of O_{target} we penalize any non-zero vector predictions for non-target class keypoints. Implications of this on model performance are discussed in section 4

Total loss for backpropagation is the simple, unweighted sum $\ell_{total} = \ell_{class} + \ell_{keypoints}$. We did not have time to explore the impact of various weighting functions for combining these two losses.

We trained our model for a maximum of 30 epochs versus 200 in the PVNet paper[6]. During training we performed basic data augmentation using random rotation, random resize crop, and colour jitter.

3.2. Voting-based Keypoint Generation

Given segmentation and keypoint output from the CNN, we proceed to generate estimates for 2D positions of keypoints using RANSAC-based voting. First, we select two pixels classified by segmentation output to be within the target class. For each keypoint k, we generate a hypothesis

$h_{k,i}$ for that keypoint which is the intersection of the 2D lines defined by the ith pixel pair, and the direction vector predictions $\tilde{v}_{k,1}$ and $\tilde{v}_{k,2}$ from the pixel pair. For each hypothesis i we calculate a voting score $w_{k,i}$ for each hypothesis over the keypoint predictions of every pixel in the class. A pixel p is considered to "vote" for a hypothesis if the direction of the vector prediction $\tilde{v}_{k,p}$ is within a certain threshold of the direction vector of from the pixel to the hypothesis $h_{k,i} - p$.

Mathematically this is formulated as:

$$vote_{k,i,p} = I\left(\frac{(h_{k,i}-p)^T}{||h_{k,i}-p||_2}\tilde{v}_{k,p} > \theta\right)$$

$$w_{k,i} = \frac{\sum_{p\epsilon O} vote_{k,i,p}}{\sum_{p\epsilon O} 1}$$

Where $p\epsilon O$ means that the pixel p belongs to the object O, I is the indicator function, $\tilde{v}_{k,p}$ is the prediction output from that pixel θ represents the closeness threshold of the vote to the hypothesis–we use 0.999 as our threshold. The outcome $w_{k,i}$ represents the share of pixels in class O that vote for hypothesis i for keypoint k. The hypothesis generation voting continues until a max number of iterations, or the worst-performing metric achieves the confidence threshold based on the RANSAC confidence condition – we use 0.99. A higher score w means that a larger share of pixels in the object are pointing toward that hypothesis, coinciding with higher confidence in the prediction

Unlike PvNet[6] we do not implement any uncertainty characterization around our keypoints.

3.3. PnP Pose Esimation

Given a set of n 3D points and the corresponding 2D keypoints, Perspective-and-Point (PnP) algorithm is used to estimate the pose(position and orientation) of the object. To estimate the 6 degrees of freedom, we need at least 3 correspondences but the PnP algorithm requires at least 4 correspondances to estimate the 6D pose uniquely. We simply use the iterative PnP algorithm implemented in OpenCV.

We did not produce variance estimates in Keypoint generation, and also did not leverage uncertainty-driven PnP as they did in the original PvNet [6], where they refines this estimate using iterative solving to minimize the Mahalanobis distance. Per the ablation studies in the original paper, this likely only contributed a small amount to the accuracy degradation we observed in our model.

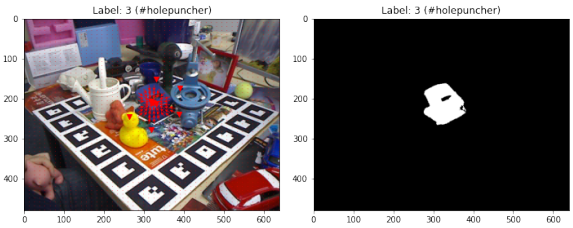

Figure 4. Example label data

4. Experiment

4.1. Data

LINEMOD [8] is a standard benchmark for 6D object pose estimation. This dataset exhibits many challenges for pose estimation: cluttered scenes, texture-less objects, and lighting condition variations. There are a total of 13 object types of interest. However, owing to limitations in training time and compute, we only trained on 5 classes of the LINEMOD dataset ("Egg", "Duck", "Cat', "Camera", and "Holepuncher").

Each sample in LINEMOD has an RGB input image, and a label with three components to label data for LINEMOD:

- **A class type** – in the dataset there is exactly one class per image, regardless of how many object types are actually in the image
- **A segmentation mask**, containing binary value for whether a pixel belongs to the class type
- **Keypoint coordinates**: 8 2D image coordinates of the 8 key points of interest.

We process the 2D image coordinates to compute a per-pixel vector of x,y coordinates describing the unit vector $v_{p,k} = \frac{k-p}{||k-p||}$ for each pixel p and keypoint k

One very important aspect of the LINEMOD dataset we obtained was that each sample image only has segment vector labels for one class type per image, despite there frequently being other "known" objects in the image that the model is trained to generate predictions for. In our current training implementation, those pixels are essentially labeled as zero (for vectors) or null (for segmentation), and as such any non-zero segmentation or keypoint predictions made by the model for objects in other classes are heavily penalized, even though they may actually be correct.

An area for further work is to explore whether these partial labels make it harder for the model to converge to good performance. It seems likely that this partial labeling would have substantial negative effect on model training which may not even be addressed by longer training. A

methods	BB8 [7]	Tekin [4]	PVNet [6]	OURS @ 5px	OURS @ 10px
ape	95.3	92.1	99.23	na	na
benchwise	80	95.06	99.81	na	na
cam	80.9	93.24	99.21	0.00	1.15
can	84.1	97.44	99.9	na	na
cat	97	97.41	99.3	0.00	0.87
driller	74.1	79.41	96.92	na	na
duck	81.2	94.65	98.02	0.41	19.18
eggbox	87.9	90.33	99.34	3.50	24.90
glue	89	96.53	98.45	na	na
holepuncher	90.5	92.86	100	0.00	0.40
iron	78.9	82.94	99.18	na	na
lamp	74.4	76.87	98.27	na	na
phone	77.6	86.07	99.42	na	na
average	83.9	90.37	99	0.78	9.30

Table 1. **LINEMOD 2D Projection** Accuracies at 5px threshold

methods	BB8 [7]	Tekin [4]	PVNet [6]	OURS @10%	OURS @ 20%
ape	27.9	21.62	43.62	na	na
benchwise	62	81.8	99.9	na	na
cam	40.1	36.57	86.86	0.00	2.69
can	48.1	68.8	95.47	na	na
cat	45.2	41.82	79.34	0.00	0.87
driller	58.6	63.51	96.43	na	na
duck	32.8	27.23	52.58	0.00	3.27
eggbox	40	69.58	99.15	1.17	12.84
glue	27	80.02	95.66	na	na
holepuncher	42.4	42.63	81.92	0.00	1.21
iron	67	74.97	98.88	na	na
lamp	39.9	71.11	99.33	na	na
phone	35.2	47.74	92.41	na	na
average	43.6	55.95	86.27	0.23	4.18

Table 2. **LINEMOD ADD(-S)** Accuracies at 10% of diameter

potential modification would be to downweight or potentially fully ignore losses from classes that are not labeled in ground truth. We ultimately did not have time to test the impact that various alternative weighting strategies would have on model performance.

4.2. Evaluation

We use the two metrics described in the original PVNet paper[6] to evaluate our model: 2D projection error [3] and average 3D distance of model points (ADD)[2].

- **2D Projection Error**. 2D projection error is the mean distance between the projections of 3D model points into the image, between the estimated and the ground truth pose. For accuracy calculations, the prediction is considered as correct if the distance is less than 5 pixels.
- **ADD and ADD-S Error.** Average Displacement Distance computes the 3D distance in keypoints between the ground truth and estimate pose, then takes the mean of this value. For accuracy calculations, an estimate is considered correct When the distance is less than 10% of the model's diameter. We approximate model diameter as the max side length of its bounding box. For symmetric objects, we use the ADD-S metric [9], where the mean distance is computed based on the closest point distance.

4.3. Results

2D Projection accuracy and ADD(-S) accuracy are presented in Table 1 and 2 respectively. After initial training, our accuracy metrics are unfortunately substantially worse than the original paper. However, we are able to generate a handful of accurate predictions. If we loosen the threshold for accuracy slightly, we see that our model is able to make reasonably close predictions. These accuracy metrics roughly correspond to qualitative results presented in Figure 6. In general we are able to produce estimates in the proximity of ground truth, but not enough to be considered correct. We occasionally have extreme outliers, which most frequently are related to poor segmentation.

One major contributor to model performance is training time. Due to compute and training limitations, we were only able to train for 30 epochs (20 epochs for cat, cam, and holepuncher) versus 200 epochs in PVNet [6]. Figure 5. shows the trend observed in both error metrics. Both metrics saw consistent improvements in error rates by the time we stopped training, and likely would have continued to improve with further training. This was especially apparent for when training 4 or more classes at once, where we found it took much longer to achieve comparable performance than when just training on 2 classes.

5. Conclusion

The goal of this project was to explore the components of a state-of-the-art CNN-based approach to pose estimation. We were able to re-implement the basic prediction pipeline described in PVNet [6] and produce pose estimations that were roughly correct, and in some cases very close to the ground truth pose.

Our results highlight the importance of training time, as well as multiple layers of refinement and improvement in order to achieve true state of the art performance. In the future, we would like to further explore some of the following

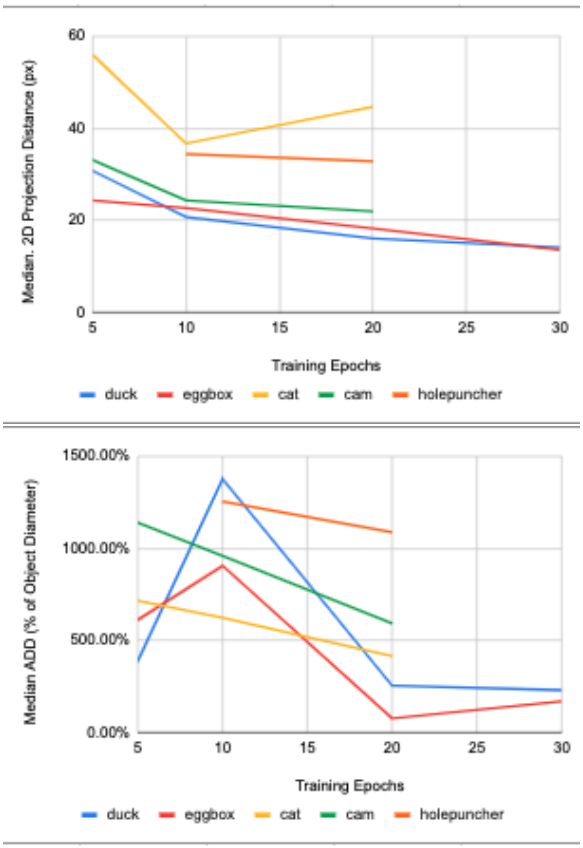

Figure 5. Error metrics by class and training duration

key factors contributing to model performance:

- Significantly longer training to 200 epochs
- Identifying and predicting surface-based keypoints instead of corner keypoints
- Implement uncertainty driven PnP which leverages the probabilistic keypoint map produced by RANSAC to produce a more refined estimate of pose
- Adjust training to ignore or downweight loss for non-target class examples, or augment provided labels to fully label all objects in class.
- Extend training using synthetically generated images
- Extending training validation using datasets with truncated or occluded images
- Supporting multiple instances per image. Unlike original PVNet, our implementation does not support classifying 6DOF pose for more than one instance of an object per image.

We would also look to explore whether different model architectures than the Resnet-18 based approach would produce comparable or better learning rates and final results than this implementation.

References

[1] Viewpoint-independent object class detection using 3d feature maps, 2008. 1

[2] Model based training, detection and pose estimation of texture-less 3d objects in heavily cluttered scenes, 2012. 5

[3] Learning 6d object pose estimation using 3d object coordinates, 2014. 1, 5

[4] S. N. S. B. Tekin and P. Fua. Real-time seamless single shot 6d object pose prediction, 2018. 5

[5] S. R. K. He, X. Zhang and J. Sun. Deep residual learning for image recognition., 2016. 2

[6] S. Peng, Y. Liu, Q. Huang, H. Bao, and X. Zhou. Pvnet: Pixel-wise voting network for 6dof pose estimation, 2018. 1, 3, 4, 5

[7] M. Rad and V. Lepetit. Bb8: A scalable, accurate, robust to partial occlusion method for predicting the 3d poses of challenging objects without using depth, 2017. 1, 5

[8] S. I. S. H. G. B. K. K. S. Hinterstoisser, V. Lepetit and N. Navab. Model based training, detection and pose estimation of texture-less 3d objects in heavily cluttered scenes, 2012. 4

[9] V. N. D. F. Yu Xiang, Tanner Schmidt. Posecnn: A convolutional neural network for 6d object pose estimation in cluttered scenes, 2017. 1, 5

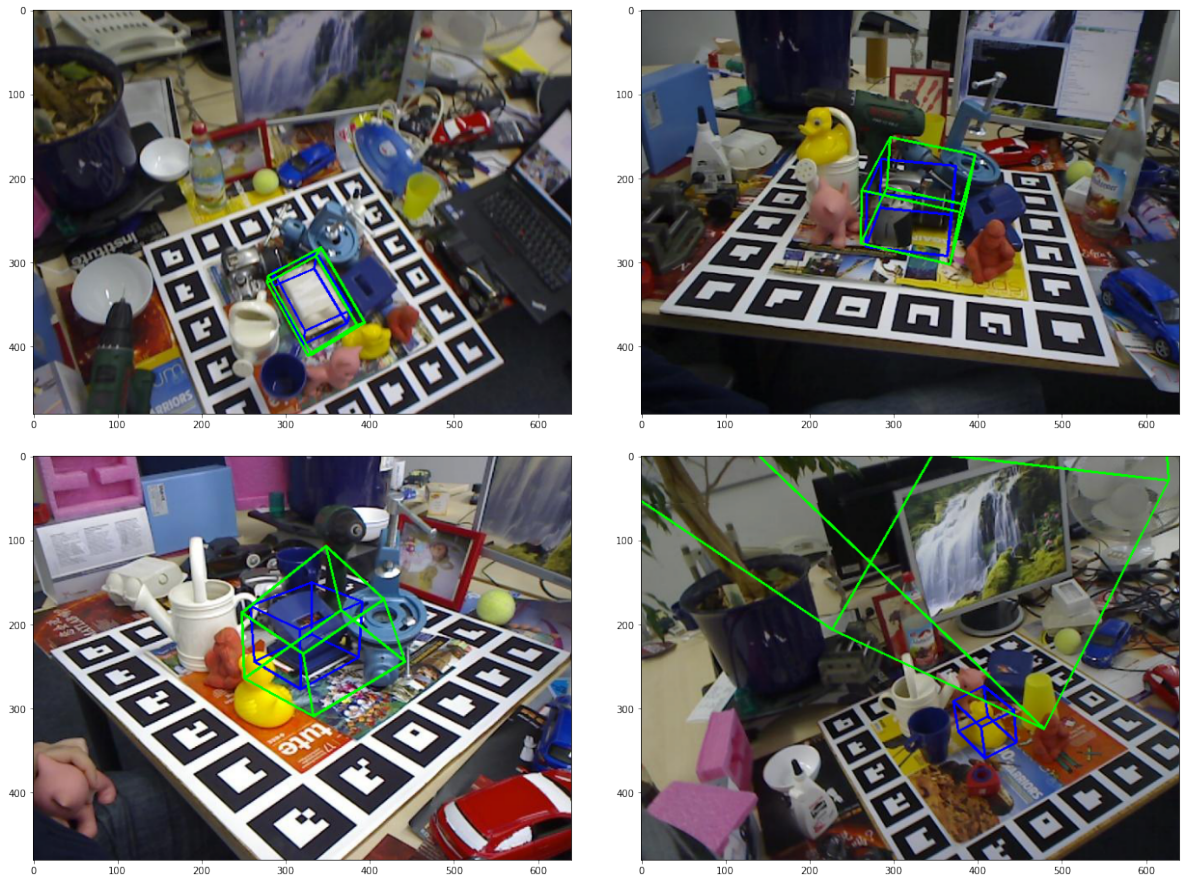

Figure 6. Bounding boxes output by our model (green), with ground truth in blue. Predictions are usually approximately correct, but there are also clear outliersn