Robust Object Pose Tracking for Robotics

Chelsea Finn Justin Fu Nopphon Siranart

***Abstract*—In this paper, we propose a neural network based approach for visual 3D object tracking, a task important for applications such as robotic manipulation and video scene understanding. We experiment with different architectures (recurrent and feedforward) and artificial data augmentation (occlusions, background, and lighting variation) to improve prediction accuracy. We also experiment with batch normalization, a technique used to both regularize and speed up neural network training.**

We found that using a recurrent architecture to take advantage of temporal correlations in the data can yield almost a 10% improvement in accuracy over a feedforward model, and can handle situations with occlusions where the feedforward model fails. Training on a harder, augmented dataset resulted a small loss in accuracy when compared to the model trained and tested on the unmodified dataset, yet we expect such training to improve robustness to real-world variation in background and lighting.

I. Introduction

Convolutional Neural Networks (CNNs) have proven to be effective in many vision tasks, most notably, object recognition [5] but also human pose estimation [7] and rigid object viewpoint classification [8]. In this paper, we explore the problem of 3D pose estimation for rigid objects held by a PR2 robot, where 3D pose is defined as 3 coordinates in 3D corresponding to 3 points on the target object (3 points such that the pose is fully specified). Such data can be obtained with a robot by moving the target object with the robot, recording the robot's joint angles, and calculating the points in 3D using the joint angles and forward kinematics model of the robot's arm.

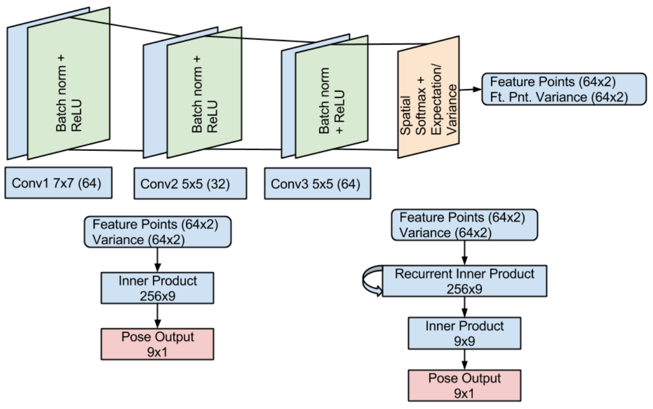
In our setting where training data is collected by a robot, the robot will typically only be trained in a single setting, yet, in the real world, need to be robust to variations in the environment. To encourage invariance to such variation, we augment the training images with artificial lighting and background variation, and then train on this augmented data.

Lastly, because sequences of images are available in a robotics setting, we investigate two approaches to utilize such temporal data: (1) using a recurrent neural network (RNN) and (2) filtering the output of the feedforward network with a Kalman filter. In our experiments, we find that a recurrent neural network architecture requires the most training time, but achieves the highest performance in tracking the pose of a target object.

II. Object Tracking with CNNs

A. Network Architecture

We performed our experiments on two different architectures, shown in Figure 1, which are based off of an architecture used for pose estimation and visuomotor policies for robotic manipulation [1].


Batch norm + ReLU
Batch norm + ReLU
Batch norm + ReLU
Spatial Softmax + Expectation/ Variance
Feature Points (64x2) Ft. Pnt. Variance (64x2)
Conv1 7x7 (64)
Conv2 5x5 (32)
Conv3 5x5 (64)
Feature Points (64x2) Variance (64x2)
Inner Product 256x9
Pose Output 9x1
Feature Points (64x2) Variance (64x2)
Recurrent Inner Product 256x9
Inner Product 9x9
Pose Output 9x1

a) Feedforward b) Recurrent

Fig. 1: A diagram of our network architecture. The top half shows the structure of the convolutional layers, which is the same for both networks. The bottom half shows the fully-connected layers, which are different for the feedforward and RNN versions of the network.

Each architecture begins with three convolutional layers, each of which is followed by a batch normalization layer and a rectifying nonlinearity $a_{cij} = max(0, z_{cij})$ for each channel c and each pixel coordinate (i, j). Note that these networks do not include any pooling in order to preserve spatial information.

The third convolutional layer contains 64 response maps with resolution 109×109. These response maps are passed through a spatial softmax function of the form $s_{cij} = e^{a_{cij}}/\sum_{i'j'} e^{a_{ci'j'}}$. Each output channel of the softmax is a probability distribution over the location of a feature in the image. To convert from this distribution to a spatial representation, the network calculates the expected image position of each feature, yielding a 2D coordinate for each channel (64x2). We also calculate the variance for each coordinate, yielding another 64x2=128 values.

The feature points are then passed into either a single 256x9 fully connected layer in the feedforward version of the network, or a 256x9 recurrent layer followed by a hyperbolic tangent non-linearity and a 9x9 recurrent layer.

The recurrent layer is defined as $h_t = g(Wx_t + Vh_{t-1} + b)$, where h_t is the output of the layer for time t, x_t is the input to the layer, g is a hyperbolic tangent, and W, V, b are learned weight matrices and biases [6].

Note that with this architecture, the model learns "feature points" in 2D image space as a result of the spatial softmax and expectation computations. We visualize these learned feature points in Figure 5.

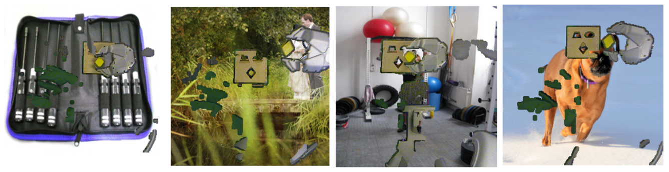


Fig. 3: Augmented training images with artificial backgrounds, used to encourage background invariance in the pose tracking neural network.

B. Training Data

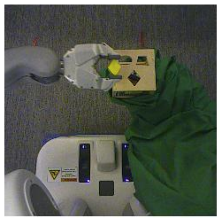


Fig. 2: An example image in the dataset

Our training data was collected via Kinect camera mounted on the head of a PR2 robot. It consisted of 240x240 RGB image sequences of the robot inserting various colored blocks (red, yellow, blue) into a target box located in 3 different positions. Each sequence was 20-frames long, and we allocated 72 sequences to the training set and 9 sequences to the test set. For the feedforward networks, the order of frames in the train and test sets was randomized, but for the recurrent network, the ordering was preserved. The ground truth for each frame was the 3D pose of the block held in the right gripper of the robot. A forward kinematics calculation using the joint angles of the robot was used to determine the position and orientation of the object. We measured accuracy by computing euclidean distance between the estimated pose and actual pose of the object.

III. Training for Robustness

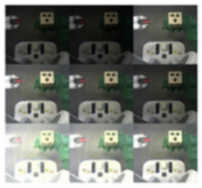
A. Training with Augmented Datasets

We additionally augmented our training dataset in various ways in order to evaluate different network architectures and encourage robustness to various scenarios, including (1) occlusions, (2) background variaiton, and (3) lighting variation.

1) Background Variation: To encourage invariance to background variation, we generated a new training dataset of images with background variation. To do so, we extracted the foreground of all of the images in the dataset, and replaced the background with a random image.

We generated binary foreground/background masks for each image in the training set by thresholding each image against the median training image and smoothing the result with basic morphological processing operations. We empirically found that most of the resulting masks correctly segmented out the background of the image, though some masks failed to due to variation in background lighting. This could be mitigated by enforcing a prior that the edges of the image are generally background pixels, and using a Markov Random Field (MRF) to segment the image. However, the foreground extraction need not be perfect because it is simply being used to generate more training images, where unsuccessful foreground extractions simply results in largely un-augmented training images.

We replaced the image backgrounds with random images sampled from ImageNet [2], scaled to match the size of the training images. Example augmented images are shown in Figure 3.



2) Lighting Variation: To encourage invariance to scene lighting, we generated a new training dataset of images with lighting variation. To do so, we converted each training image to the L*a*b* color space and offset the L channel of each pixel independently by uniformly randomly chosen number between -30 and 30. Some example images are shown in to the right.

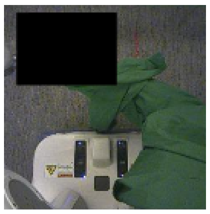


Fig. 4: Occluded gripper and target.

3) Occlusions: We explore the ability of our network architecture to adapt to occlusions of the target object. For each 20-frame sequence in the train and test set, we selected a 5-frame interval starting at [5, 15] uniformly at random, and blacked out a fixed region in the image which contained the gripper and target for those frames.

B. Batch Normalization

Batch normalization [3] is an architecture modification that acts as a regularizer and allows usage of higher learning rates for faster training. For a fully connected layer, the batch normalization layer first whitens the data within a batch. For a layer with input $x = (x_1, ...x_n)$, each x_i is transformed by:

$$\hat{x_i} = \frac{x_i - E[x_i]}{\sqrt{Var[x_i]}}$$

Then, the layer applies a learned scale and shift parameter.

$$y_i = \gamma_i \hat{x_i} + \beta_i$$

Note that if $\gamma_i = \sqrt{Var[x_i]}$ and $\beta_i = E[x_i]$, then the original values would be recovered, so the network loses no representational power.

For a convolutional layer, batch normalization is applied not only across all elements of the minibatch, but also across spatial location in the response maps. The parameters γ, β are learned 1 per channel. This way, the same operation is applied regardless of spatial location, preserving the convolution.

C. Feature Point Slowness

While training these CNN architectures, we observed that the many of the learned feature points jumped around the scene across subsequent time steps. In order to encourage temporal consistency in the feature points, we trained the CNN outlined in Section II-A with an additional loss term, which penalizes the mean euclidean distance of these feature points between two consecutive frames. This penalty encourages the model to learn feature points that move slowly from one frame to the next. Videos of the learned feature points with and without this penalty can be found at http://tinyurl.com/cs280-finalproj

IV. Experimental Results

TABLE I: Pose Estimation Results

Network	Training Dataset	Validation Accuracy (cm)
FF	Unaugmented	0.944
RNN	Unaugmented	0.862
FF + Kalman	Unaugmented	0.939
FF + Slowness	Unaugmented	1.04
FF	Backgrounds	1.05
FF	Lighting	1.00
FF + LRN	Lighting	0.95

TABLE II: Results (On occluded data)

Network	Validation Accuracy (cm)
RNN	1.35
FF	3.76

A. Feature Point Visualization

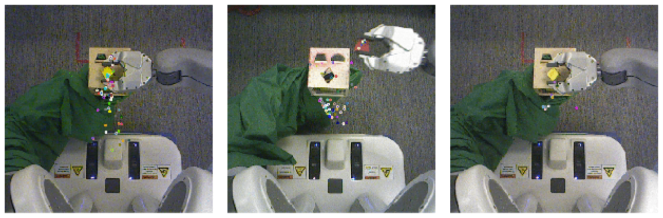


Fig. 5: Feature points learned by the neural networks. From left to right: CNN, RNN, and CNN with feature point slowness. Note that the slowness penalty results in feature points that are largely clustered in one location with only a few feature points that move with the target object.

In Figure 5, we compare the feature points learned by models trained for pose prediction including CNN, RNN and CNN with feature point slowness. Each color dot represents the a feature point as a result of Softmax layer. There are 64 feature points in total. Clearly from the video, the points from RNN and those with slowness implemented are moving more slowly and smoothly from one frame to another and tracking on task-relevant objects much better than those from CNN. Although CNN and RNN models find more feature points on the images, the model with slowness implemented have more feature points on task-relevant objects and the rest are centered at the origin.

B. Batch Normalization

We found that using a higher learning rate in conjunction with batch normalization can significantly speed up convergence times. For our experiments, we use an initial learning rate that is 6 times higher (0.03 vs 0.005) with batch normalization versus without. We plotted our experiments in Figure 6, where the step size schedule is identical for all trials. If stopped after 10,000 iterations of SGD, both networks trained with batch normalization achieved higher accuracy than those

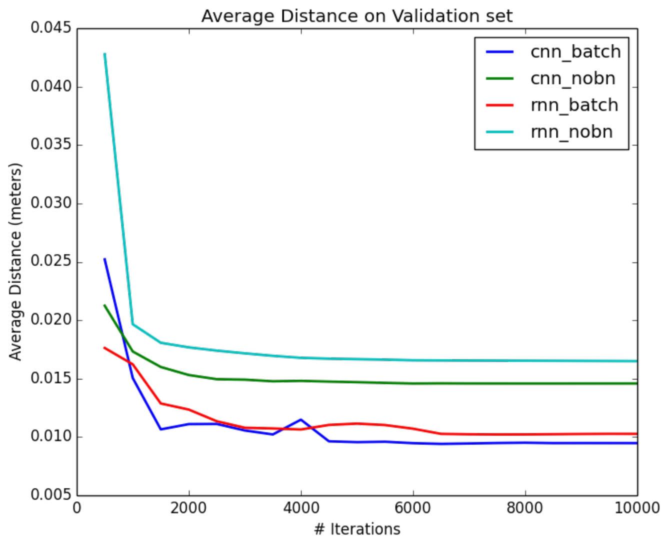


Fig. 6: Feedforward CNN and RNN performance with and without batch normalization.

without, and the feedforward architecture outperformed the RNN.

We hypothesize that the RNN could perform better using a GPU with more memory such that more than one sequence of images could be included in a single mini-batch. In our RNN optimization, each minibatch consisted of images from a single trial, with the same block color and target position. This affects the performance of stochastic gradient descent, in which the batch is ideally drawn uniformly from the dataset, and has ramifications in batch normalization for the same reason.

C. Analysis of the recurrent network

To analyze the performance of the recurrent network, we consider two scenarios.

1) *Performance on un-augmented data:*
Giving both networks time to train until convergence, both networks eventually achieved errors of under 1 cm (see table I), with the RNN performing slightly better than the feedforward (FF) model.
In this setting, we also compare to a Kalman Filter. We trained a 2nd-order, time-invariant Kalman filter over the outputs of the feedforward network. Combined with the feedforward model, this set-up enables use of the sequential image information providing a fair comparison to the RNN model.
We found that although the output is smoothed, the Kalman filter only marginally improved the performance of the feedforward network. This is possibly because the neural network error is not well modeled by Gaussian noise that is independent of the input data, and suggests that the RNN architecture is doing more than simple filtering.
2) *Performance on occluded data:*
When both networks were trained on the dataset with occlusions, the RNN outperformed the feedforward net-

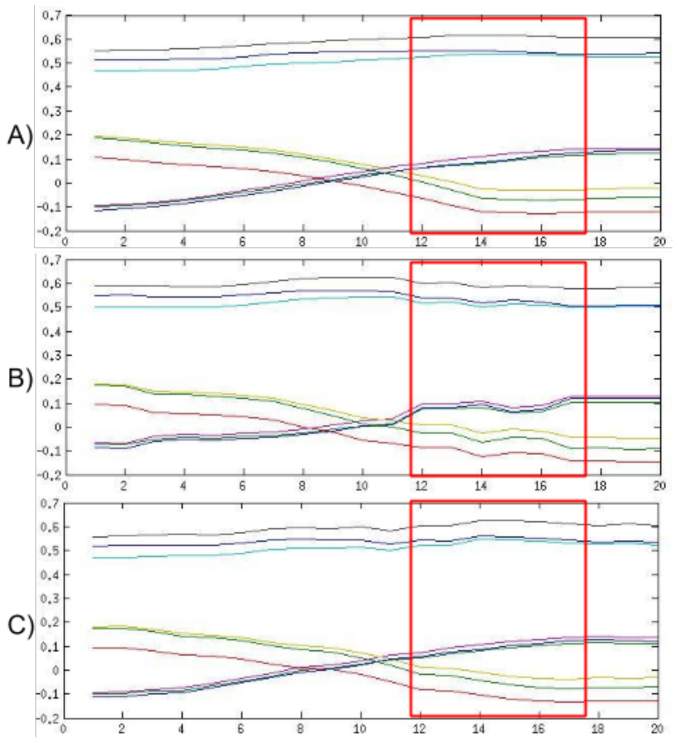


Fig. 7: Pose predictions on a 20-frame validation set trial for A) the ground truth, B) feedforward CNN, and C) RNN. The occluded frames (12-17) are outlined in red.

work by more than 2cm average accuracy (see table II). We found that the ability of the RNN to hold state over time greatly increased its ability to make accurate predictions when it could only see the environment and not the target or object.

An example plot of the network predictions are shown in Figure 7. In the figure, there is a visible ”discontinuity” in the feedforward predictions when the occlusion occurs, whereas the RNN predictions are smooth over the interval, as if no occlusion had occurred.

D. *Implementation and Computational Performance*

Our neural network training was implemented using the Caffe [4] deep learning library on an Nvidia GTX 980 GPU. We optimized with stochastic gradient descent until convergence, which took just under 1 hour of training time for the feedforward network and 2 hours for the recurrent network.

V. Future Work

In the immediate future, we hope to investigate the benefits of data augmentation with real-world scene variation, as well as experiment with different objects being held.

An additional area of interest is exploring how training data collected using the robot could be used to learn object pose trackers when the object is *not* held by the robot gripper. More broadly, the robot’s actions and movements could provide supervision for learning useful visual features.

References

[1] S. Levine C. Finn T. Darrell and P. Abbeel. End-to-end training of deep visuomotor policies. 2015.

[2] J. Deng, W. Dong, R. Socher, L. Li, K. Li, and L. Fei-Fei. ImageNet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition (CVPR)*, 2009.

[3] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. 2015.

[4] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*, 2014.

[5] A. Krizhevsky, I. Sutskever, and G. Hinton. ImageNet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems (NIPS)*. 2012.

[6] J. Donahue L. Hendricks S. Guadarrama M. Rohrbach S. Venugopalan K. Saenko and T. Darrell. Long-term recurrent convolutional networks for visual recognition and description. 2015.

[7] J. J. Tompson, A. Jain, Y. LeCun, and C. Bregler. Joint training of a convolutional network and a graphical model for human pose estimation. In *Advances in Neural Information Processing Systems (NIPS)*, 2014.

[8] Shubham Tulsiani and Jitendra Malik. Viewpoints and keypoints. *CoRR*, abs/1411.6067, 2014. URL http://arxiv.org/abs/1411.6067.