Towards Learning to Perceive and Reason About Liquids

Connor Schenck, Dieter Fox
University of Washington

Abstract. Recent advances in AI and robotics have claimed many incredible results with deep learning, yet no work to date has applied deep learning to the problem of liquid perception and reasoning. In this paper, we apply fully-convolutional deep neural networks to the tasks of detecting and tracking liquids. We evaluate three models: a single-frame network, multi-frame network, and a LSTM recurrent network. Our results show that the best liquid detection results are achieved when aggregating data over multiple frames and that the LSTM network outperforms the other two in both tasks. This suggests that LSTM-based neural networks have the potential to be a key component for enabling robots to handle liquids using robust, closed-loop controllers.

Keywords: Robot perception, deep learning, liquids, manipulation

1 Introduction

To robustly handle liquids, such as pouring a certain amount of water into a bowl, a robot must be able to perceive and reason about liquids in a way that allows for closed-loop control. Liquids present many challenges compared to solid objects. For example, liquids can not be interacted with directly by a robot, instead the robot must use a tool or container; often containers containing some amount of liquid are opaque, obstructing the robot's view of the liquid and forcing it to remember the liquid in the container, rather than re-perceiving it at each timestep; and finally liquids are frequently transparent, making simply distinguishing them from the background a difficult task. Taken together, these challenges make perceiving and manipulating liquids highly non-trivial.

Recent advances in deep learning have enabled a leap in performance not only on visual recognition tasks, but also in areas ranging from playing Atari games [1] to end-to-end policy training in robotics [2]. In this paper, we investigate how deep learning techniques can be used for perceiving liquids during pouring tasks. We develop a method for generating large amounts of labeled pouring data for training and testing using a realistic liquid simulation and rendering engine, which we use to generate a data set with 10,122 pouring sequences, each 15 seconds long, for a total of 2,531 minutes of video or over 4.5 million labeled images. Using this dataset, we evaluate multiple deep learning network architectures on the tasks of detecting liquid in an image and tracking the location of liquid even when occluded. Our results show that deep networks able to detect and track liquid in a simulated environment with a reasonable degree of robustness. We also have preliminary results that show that these networks perform well in real environments.

2 Related Work

To the best of our knowledge, no prior work has investigated directly perceiving and reasoning about liquids. Existing work relating to liquids either uses coarse simulations that are disconnected to real liquid perception and dynamics [3,4] or constrained task spaces that bypass the need to perceive or reason directly about liquids [5,6,7,8,9]. While some of this work has dealt with pouring, none of it has attempted to directly perceive liquids from raw sensory data. In contrast, in this work we directly approach this problem.

Similarly, Rankin *et al.* [10,11] investigated ways to detect pools of water from an unmanned ground vehicle navigating rough terrain. They detected water based on simple color features or sky reflections, and didn't reason about the dynamics of the water, instead treating it as a static obstacle. Griffith *et al.* [12] learned to categorize objects based on their interactions with running water, although the robot did not detect or reason about the water itself, rather it used the water as a means to learn about the objects. In contrast to [12], we use vision to directly detect the liquid itself, and unlike [10,11], we treat the liquid as dynamic and reason about it.

In order to perceive liquids at the pixel level, we make use of fully-convolutional neural networks (FCN). FCNs have been successfully applied to the task of image segmentation in the past [13,14,15] and are a natural fit for pixel-wise classification. In addition to FCNs, we utilize long short-term memory (LSTM) [16] recurrent cells to reason about the temporal evolution of liquids. LSTMs are preferable over more standard recurrent networks for long-term memory as they overcome many of the numerical issues during training such as exploding gradients [17]. LSTM-based CNNs have been successfully applied to many temporal memory tasks by previous work [18,15], and in fact LSTMs have even been combined with FCNs by replacing the standard fully-connected layers of their LSTMs with 1×1 convolution layers [15]. We use a similar method in this paper.

3 Methodology

In order to train neural networks to perceive and reason about liquids, we must first have labeled data to train on. Getting pixel-wise labels for real-world data can be difficult, so in this paper we opt to use a realistic liquid simulator. In this way we can acquire ground truth pixel labels while generating images that appear as realistic as possible. We train three different types of convolutional neural networks (CNNs) on this generated data to detect and track the liquid: single-frame CNN, multi-frame CNN, and LSTM-CNN.

3.1 Data Generation

We generate data using the 3D-modeling application Blender [19] and the library El'Beem for liquid simulation, which is based on the lattice-Boltzmann method for efficient, physically accurate liquid simulations [20]. We separate the data generation process into two steps: simulation and rendering. During simulation, the liquid simulator calculates the trajectory of the surface mesh of the liquid as the cup pours the liquid into the bowl. We vary 4 variables during simulation: the type of cup (cup, bottle, mug), the type of bowl (bowl, dog dish, fruit bowl), the

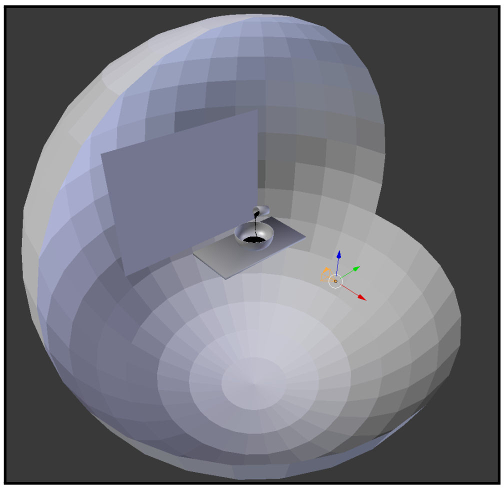


Fig. 1: The setup used to simulate and render liquid sequences. The objects are shown here textureless for clarity. The sphere surrounding all the objects has been cut away to allow viewing of the objects inside. The orange shape represents the camera's viewpoint, and the flat plane across the table from it is the plane on which the video sequence is rendered. Note that this plane is sized to exactly fill the camera's view frustum. The background sphere is not directly visible by the camera and is used primarily to compute realistic reflections.

initial amount of liquid (30% full, 60% full, 90% full), and the pouring trajectory (slow, fast, partial), for a total of 81 simulations. Each simulation lasts exactly 15 seconds for a total of 450 frames (30 frames per second).

Next we render each simulation. We separate simulation from rendering because it allows us to vary other variables that don't affect the trajectory of the liquid mesh (e.g., camera viewpoint), which provides a significant speedup as liquid simulation is much more computationally intensive than rendering. In order to approximate realistic reflections, we mapped a 3D photo sphere image taken in our lab to the inside of a sphere, which we place in the scene surrounding all the objects. To prevent overfitting to a static background, we also add a plane in the image in front of the camera and behind the objects that plays a video of activity in our lab that approximately matches with that location in the background sphere. This setup is shown in Fig. 1. The liquid is always rendered as 100% transparent, with only reflections, refractions, and specularities differentiating it from the background. For each simulation, we vary 6 variables: camera viewpoint (48 preset viewpoints), background video (8 videos), cup and bowl textures (6 textures each), liquid reflectivity (normal, none), and liquid index-of-refraction (air-like, low-water, normal-water). The 48 camera viewpoints were generated by varying the camera elevation (level with the table and looking down at a 45 degree angle), camera distance (8m, 10m, and 12m), and the camera azimuth (the 8 points of the compass, with north, southwest, and south shown in the top, middle, and bottoms rows of Fig. 2) respectively. We also generate negative examples without liquid. In total, this yields 165,888 possible renders for each

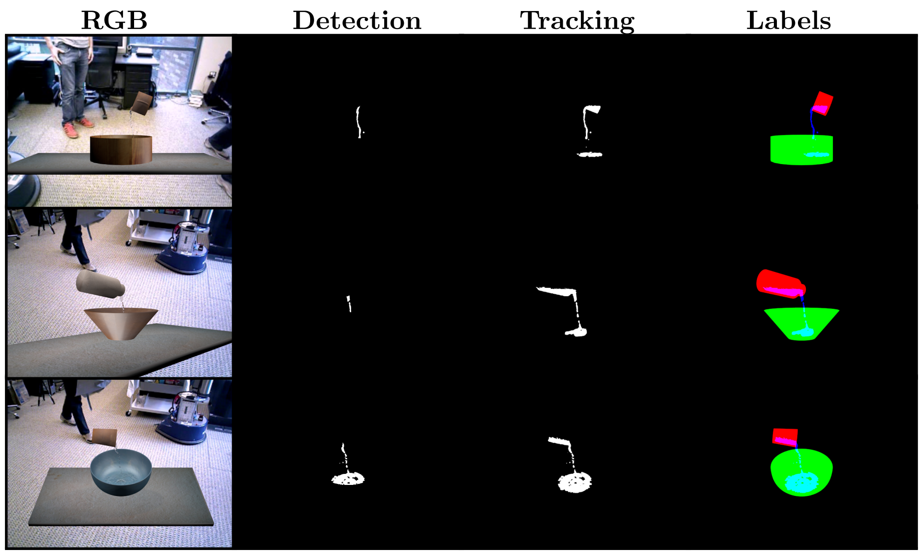


Fig. 2: Examples of frames rendered by our data generation algorithm. The left column is the raw RGB images generated by the renderer; the center-left column shows the ground truth liquid location for detection; the center-right column shows the ground truth liquid location for tracking; the right column shows the ground truth labeling output by the simulator.

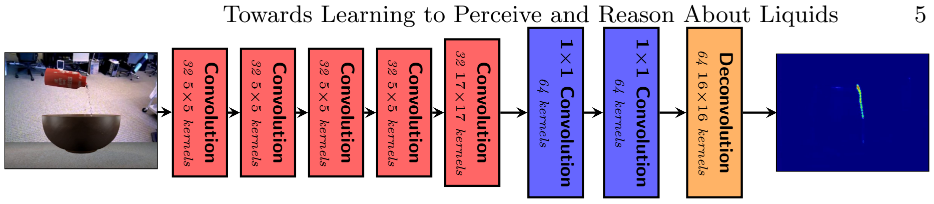
simulation. It is infeasible to render them all, so we randomly sample variable values to render.

The labels are generated for each object (liquid, cup, bowl) as follows. First, all other objects in the scene are set to render as invisible. Next, the material for the object is set to render as a specific, solid color, ignoring lighting. The sequence is then rendered, yielding a class label for the object for each pixel. An example of labeled data (right column) and its corresponding rendered image (left column) is shown in Fig. 2. The cup, bowl, and liquid are rendered as red, green and blue respectively. Note that this method allows each pixel to have multiple labels, e.g., some of the pixels in the cup are labeled as both cup and liquid (magenta in the right column of Fig. 2). To determine which of the objects, if any, is visible at each pixel, we render the sequence once more with all objects set to render as their respective colors, and we use the alpha channel in the ground truth images to encode the visible class label.

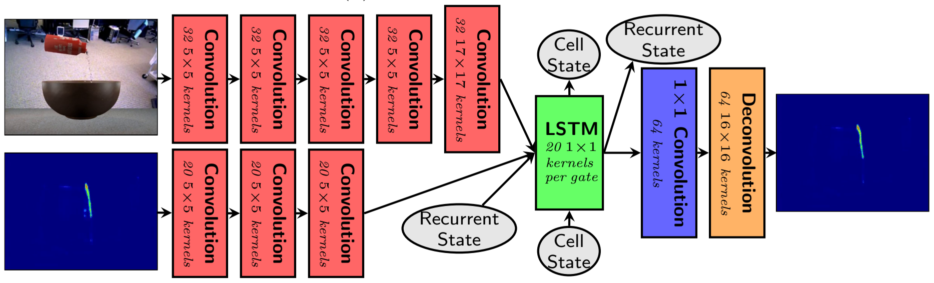
To evaluate our learning architectures, we generated 10,122 pouring sequences by randomly selecting render variables as described above as well as generating negative sequences (i.e., sequences without any water), for a total of 4,554,900 training images. Both the model files generated by Blender and the rendered images for the entire dataset are available for download at the following link: `http://rse-lab.cs.washington.edu/lpd/`.

3.2 Network Architecture

We test three network layouts for the tasks of detecting and tracking liquids: CNN, MF-CNN, and LSTM-CNN. All of our networks are fully-convolutional



(a) Single-frame CNN



(b) LSTM-CNN

Fig. 3: Layout of the single-frame and LSTM networks. Each of the red **Convolution** layers is followed by a max pooling layer and a rectified linear layer. The max pooling layers have the same kernel size as the convolution layer they follow, and the first two convolution layers in each network have a stride of 2 (all others and all convolution layers have a stride of 1). Each of the blue **1×1 Convolution** layers is followed by a rectified linear layer. Refer to figure 1 of [17] for more details on the LSTM layer.

[13], that is, there are no fully-connected layers. In place of fully-connected layers used in more standard CNNs, we use 1×1 convolutional layers, which have a similar effect but prevent the explosion of parameters that normally occurs. We use the Caffe deep learning framework [21] to implement our networks[1].

CNN The first layout is a standard convolutional neural network (CNN). It takes in an image and outputs probabilities for each class label at each pixel. It has a fixed number of convolutional layers, each followed by a rectified linear layer and a max pooling layer. In place of fully-connected layers, we use two 1×1 convolutional layers, each followed by a rectified linear layer. The last layer of the network is a deconvolutional layer that upsamples the output of the 1×1 convolutional layers to be the same size as the input image. This network is shown in Fig. 3a.

MF-CNN The second layout is a multi-frame CNN. Instead of taking in a single frame, it takes as input multiple consecutive frames and predicts the probability of each class label for each pixel at the last frame. It is similar to the single-frame CNN network shown in Fig. 3a except each frame is convolved independently through the first 5 convolution layers, and then the output for each frame is concatenated together channel-wise. This is fed to the two 1×1 convolutional layers, each followed by a rectified linear layer, and finally a deconvolutional layer. We fix the number of input frames for this layout to 32 for this paper, i.e., approximately 1 second's worth of data (30 frames per second), which we em-

[1] The network structure files (prototxt) can be found on our project page at `http://rse-lab.cs.washington.edu/projects/liquids/`

pirically determined strikes the best balance between window size and memory utilization.

LSTM-CNN The third layout is similar to the single frame CNN layout, with the first 1×1 convolutional layer replaced with a LSTM layer (see figure 1 of [17] for a detailed layout of the LSTM layer). We replace the fully-connected layers of a standard LSTM with 1×1 convolutional layers. The LSTM takes as recurrent input the cell state from the previous timestep, its output from the previous timestep, and the output of the network from the previous timestep processed through 3 convolutional layers (each followed by a rectified linear and max pooling layer). During training, when unrolling the LSTM-CNN, we initialize this last recurrent input with the ground truth at the first timestep, but during testing we use the standard recurrent network technique of initializing it with all zeros. Fig. 3b shows the layout of the LSTM-CNN.

4 Evaluation

We evaluated our networks on 4 experiments: fixed-viewpoint detection, multi-viewpoint detection, fixed-viewpoint tracking, and combined detection & tracking. We define the detection task as, given raw color images, determine where the *visible* liquid in the images is. We define the tracking task as, given segmented images (i.e., images that have already been run through a detector), determine where *all* liquid (visible and occluded) is in the image. Intuitively, detection corresponds to perceiving the liquid, while tracking corresponds to reasoning about where the liquid is given what is (and has been) visible.

Every network was trained using the mini-batch gradient descent method Adam [22] with a learning rate of 0.0001 and default momentum values. Each network was trained for 61,000 iterations, at which point performance tended to plateau. All single-frame networks were trained using a batch size of 32; all multi-frame networks with a window of 32 and batch size of 1; and all LSTM networks with a batch size of 5. For all experiments except the third (fixed-viewpoint tracking), the input images were scaled to 400×300 resolution. The error signal was computed using the softmax with loss layer built into Caffe [21].

We empirically determined, however, that naively training a network in this setup results in it predicting no liquid present in any scene at all due to the significant positive-negative class imbalance (most of the pixels in each image are non-liquid pixels). To counteract this we employed two strategies. The first was to pre-train the network on 160×160 crops of the image around liquid pixels. Since our networks are fully-convolutional, they can have variable sized inputs and outputs, which means a network pre-trained in this manner can be immediately trained on full images without needing any modification. The second strategy was to weight the gradients from the error signal based on the class of the ground truth pixel: 1.0 for positive pixels and 0.1 for negative pixels. This decreases the effect of the non-liquid pixels and prevents the network from predicting no liquid in the scene.

We report the precision and recall of each network on a hold-out test set, evaluated on pixel-wise classifications. We also report the precision and recall for various amounts of "slack," i.e., we count a pixel labeled as liquid correct if it is within n pixels of a ground truth liquid pixel, where n is the amount of slack. This better evaluates the network in cases where it's predictions are only a few pixels off, which is a relatively small error given the resolution of the images.

4.1 Experiment 1: Fixed-Viewpoint Detection

We evaluated all three network types on a fixed-viewpoint detection task. We define fixed-viewpoint in this context to mean data generated as described in Section 3.1 for which the camera elevation is level with the table and the azimuth is either north (as shown in the top row of Fig. 2) or south (180 degrees opposite). The networks were given the full rendered RGB image as input (similar to the left column in Fig. 2) and the output was a classification at each pixel as liquid or not liquid. To counteract the class imbalance, we employed visible liquid image crop pre-training for each network (we initialized the image crop LSTM-CNN with the trained weights of the image crop single-frame CNN). We then trained the final network for each type on full images initializing it with the weights of the image crop network. During training, the LSTM-CNN was unrolled for 32 timesteps.

4.2 Experiment 2: Multi-Viewpoint Detection

For the second experiment, we expanded the data used to include all 48 viewpoints, presenting a non-trivial increase in difficulty for the networks. Our goal was to test the generalizability of the networks across a much wider variation in viewpoints. For this reason, we focused only on testing the best performing network, the LSTM-CNN (see Section 5.1 for results from experiment 1). Also to test generalizability, we only trained the network on a subset of the 48 viewpoints, and tested on the remaining. We used all data generated using the 8m and 12m camera viewpoint distances for training and data generated using the 10m camera distance for testing. We also employed the gradient weighting scheme described above to counteract the class imbalance. The LSTM-CNN was trained in the same manner as in experiment 1.

4.3 Experiment 3: Fixed-Viewpoint Tracking

For tracking only, the networks were given pre-segmented input images, with the goal being to track the liquid when it is not visible. An example of this input is shown in the first row of the right column from Fig. 2, with the exception that the occluded liquid (magenta and cyan) were not shown. Because these input images are more structured, we lowered the resolution to 130×100. The output was the pixel-wise classification of liquid or not liquid, including pixels where the liquid was occluded by other objects in the scene. During training, the LSTM-CNN was unrolled for 160 timesteps. We reduced the number of initial convolution layers on the input from 5 to 3 for each of the three networks. Due to the structured nature of the input, each network was trained directly on full images with Gaussian-random weight initialization. We used the data from the same viewpoints (level with the table and azimuth at north or south) as in experiment 1.

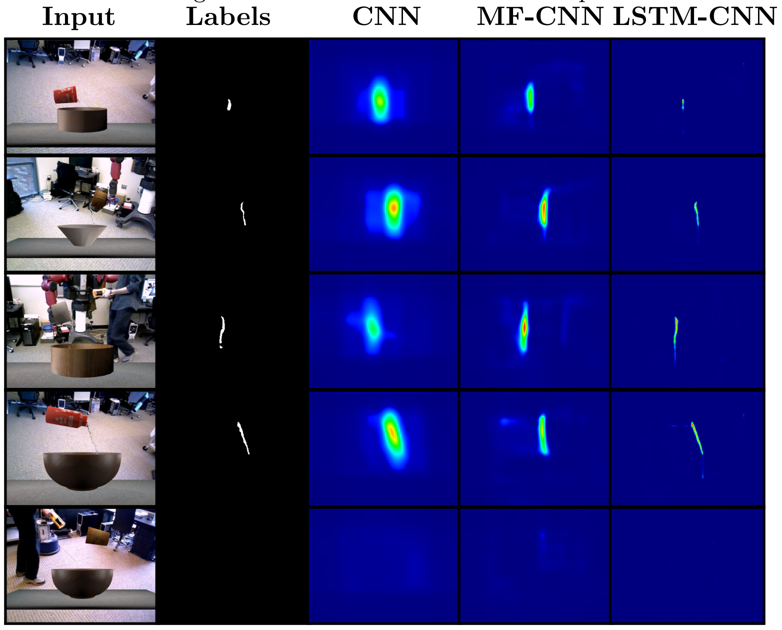


Fig. 4: Qualitative fixed-viewpoint liquid detection results. The Input column is the input to the networks, the Labels column is the ground truth labeling of each pixel as liquid or not liquid, and the CNN, MF-CNN, and LSTM-CNN columns show a heatmap of the prediction of each network for each of the input frames. 5 sequences were randomly selected from our training set, and the frame with the most liquid pixels was picked for display here, with the exception of the last row, which shows how the networks perform when there is no liquid present.

4.4 Experiment 4: Combined Detection & Tracking

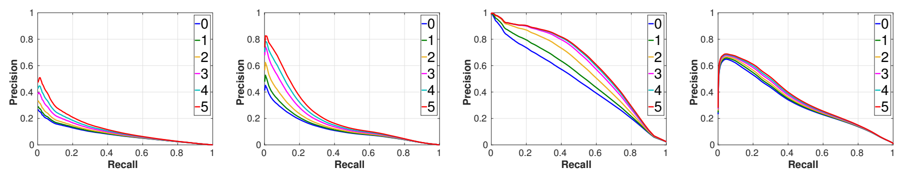
For the last experiment, we combine detection and tracking into a single task, i.e., given raw color images, determine where *all* liquid in the scene is (visible and occluded). Our goal is to determine if it is possible to do both tasks with one network, and for this reason, we evaluate only the LSTM-CNN. We initialized the network with the weights of the trained LSTM-CNN from experiment 1 and trained it on full images. As in experiment 2, we employed the gradient weighting scheme described above to counteract the class imbalance. We used the data from the same viewpoints as in experiment 1 and 3.

5 Results

5.1 Fixed-Viewpoint Detection

Fig. 4 shows qualitative results for the three networks on the liquid detection task[2]. The frames in this figure were randomly selected from the training set, and it is clear from the figure that all three networks detect the liquid at least to some degree. Figures 5a to 5c show a quantitative comparison between the three networks. As expected, the multi-frame CNN outperforms the single-frame. Surprisingly, the LSTM-CNN performs much better than both by a significant margin. These results strongly suggest that detecting transparent liquid must be done over a series of frames, rather than a single frame.

[2] Video of the full sequences at `https://youtu.be/m5z0aFZgEX8`



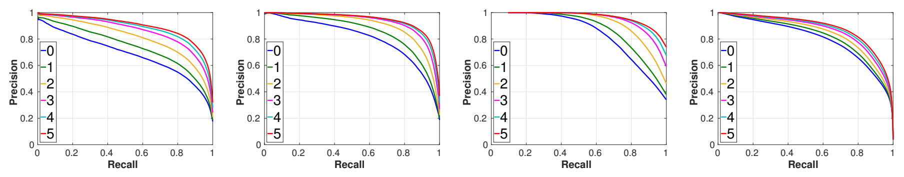
(a) CNN (b) MF-CNN (c) LSTM-CNN (d) MV LSTM-CNN

Fig. 5: Quantitative fixed- and multi-viewpoint liquid detection results. The graphs indicate the precision and recall for each of the three networks on fixed-viewpoint detection and the LSTM-CNN on multi-viewpoint detection. The colored lines indicate the variation in the number of slack pixels we allowed for prediction, i.e., how many pixels a positive classification could be away from a positive ground truth labeling and still be counted as correct.

5.2 Multi-Viewpoint Detection

Fig. 5d shows the results from multi-viewpoint detection for the LSTM-CNN. As expected, the 8-fold increase in number of viewpoints leads to lower performance as compared to Fig. 5c, but overall it is clearly still able to detect the liquid reasonably well. Interestingly, there is less spread between the various levels of slack than in Fig. 5c, meaning the network benefits less from increased slack, suggesting that it is less precise than the fixed-view LSTM-CNN, which makes sense given the much larger variation in viewpoints.

5.3 Fixed-Viewpoint Tracking



(a) CNN (b) MF-CNN (c) LSTM-CNN (d) DT LSTM-CNN

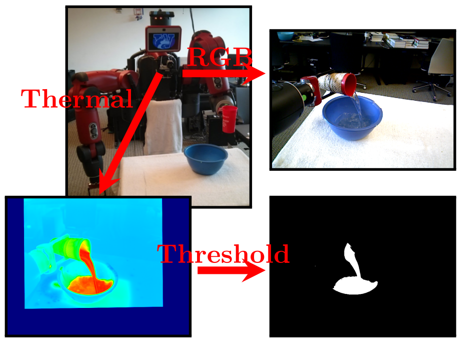
Fig. 6: Fixed-viewpoint liquid tracking and combined detection & tracking results. Similar to Fig. 5, the graphs indicate the precision and recall for each of the three networks and the colored lines indicate the variation in the number of slack pixels we allowed for prediction.

For tracking, we evaluated the performance of the networks on locating both visible and invisible liquid, given segmented input (i.e., each pixel classified as liquid, cup, bowl, or background). Because the viewpoint was fixed level with the bowl, the only visible liquid the network was given was liquid as it passed from cup to bowl. Figures 6a to 6c show the performance of each of the three networks. As expected, the LSTM-CNN has the best performance. Interestingly, the multi-frame CNN performs better than expected, given that it only sees approximately 1 second's worth of data and has no memory capability.

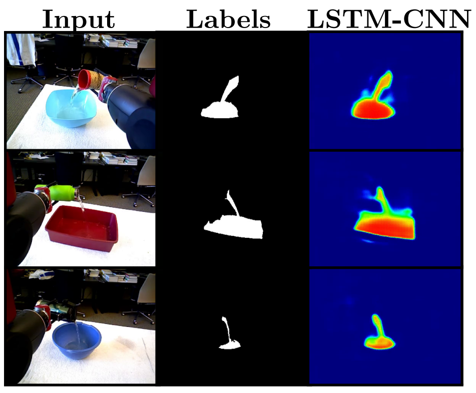
5.4 Combined Detection & Tracking

Fig. 6d shows the results of combined detection and tracking for the LSTM-CNN. Given a raw color image, the network predicted where both the visible and occluded liquid was. Comparing this to the rest of Fig. 6, it is clear that the network was able to do quite well, despite using raw, unstructured input, unlike the other networks in that figure. This strongly suggests that LSTM-CNNs are best suited not only for detecting liquids, but also tracking them.

5.5 Preliminary Real Robot Results



(a) Robot setup for collecting data.



(b) Example output of the LSTM-CNN.

Fig. 7: The left figure shows our robot, it's attached thermal and RGB cameras, and example output of each camera. The right figure shows the output of a LSTM-CNN trained on data collected on the real robot. The Input column is the input to the network, the Labels column is the ground truth labeling of each pixel as liquid or not liquid, and the LSTM-CNN column shows a heatmap of the prediction of the network for each of the input frames.

Fig. 7b shows qualitative results of the LSTM-CNN trained on a small dataset collected on a real robot in our lab[3]. We used a thermal infrared camera calibrated to our RGB camera in combination with heated water to acquire ground truth labeling for data collected using a real robot. The advantage of this method is that heated water appears identical to room temperature water on a standard color camera, but is easily distinguishable on a thermal camera. This allows us to label the "hot" pixels as liquid and all other pixels as not liquid. Fig. 7a shows our robot setup with the thermal and RGB cameras. It is clear from Fig. 7b that our methods, to at least a limited degree, apply to real world data and not just data generated by a liquid simulator.

6 Discussion & Conclusion

The results in Section 5 show that it is possible for deep learning to detect and track liquids in a scene, both independently and combined, and also over a wide variation in viewpoints. Unlike prior work on image segmentation, these results clearly show that single images are not sufficient to reliably perceive liquids. Intuitively, this makes sense, as a transparent liquid can only be perceived through its refractions, reflections, and specularities, which vary significantly from frame to frame, thus necessitating aggregating information over multiple frames. We

[3] Full video of results at `https://youtu.be/4pbjSqg5zfQ`

also found that LSTM-based CNNs are best suited to not only aggregate this information, but also to track the liquid as it moves between containers. LSTMs work best, due to not only their ability to perform short term data integration (just like the MF-CNN), but also to remember states, which is crucial for tracking the presence of liquids even when they're invisible.

From the results shown in Fig. 4 and in the video[4], it is clear that the LSTM CNN can at least roughly detect and track liquids. Nevertheless, unlike the task of image segmentation, our ultimate goal is not to perfectly estimate the potential location of liquids, but to perceive and reason about the liquid such that it is possible to manipulate it using raw sensory data. For this, a rough sense of where the liquid is in a scene and how it is moving might suffice. Neural networks, then, have the potential to be a key component for enabling robots to handle liquids using robust, closed-loop controllers.

7 Future Work

We are currently on expanding the real robot results from Section 5.5. As stated in Section 3, it can be difficult to get the ground truth pixel labels for real data, which is why we chose to use a realistic liquid simulator in this paper. However, our method of combing a thermal camera with heated water to get the ground truth makes it feasible to apply the techniques in this paper to data collected on a real robot. For future work we plan to collect more data on the real robot using this technique and do a thorough analysis of the results.

Another avenue of future work we are currently pursuing is extending these techniques to control problems. The results here clearly show that deep neural networks can effectively be used to detect and, to some extent at least, reason about liquids. The next logical step is to utilize neural networks to manipulate liquids via a robot. One potential algorithm to accomplish this is Guided Policy Search (GPS) [23], which learns a control policy for a task from raw sensory data. The advantage of an algorithm like GPS is that it works well on high-dimensional sensory input where collecting large amounts of data may be infeasible (as is often the case on a real robotic system). In future work we plan to apply a similar algorithm to the problem of robotic liquid control from raw sensory data.

8 Acknowledgments

This work was funded in part by the National Science Foundation under contract number NSF-NRI-1525251 and by the Intel Science and Technology Center for Pervasive Computing (ISTC-PC).

References

1. Guo, X., Singh, S., Lee, H., Lewis, R.L., Wang, X.: Deep learning for real-time atari game play using offline monte-carlo tree search planning. In: NIPS. (2014) 3338–3346
2. Levine, S., Finn, C., Darrell, T., Abbeel, P.: End-to-end training of deep visuomotor policies. arXiv preprint arXiv:1504.00702 (2015)
3. Kunze, L., Beetz, M.: Envisioning the qualitative effects of robot manipulation actions using simulation-based projections. Artificial Intelligence (2015)

[4] Video of the full sequences at `https://youtu.be/m5z0aFZgEX8`

4. Yamaguchi, A., Atkeson, C.G.: Differential dynamic programming with temporally decomposed dynamics. In: Humanoids. (2015) 696–703
5. Langsfeld, J., Kaipa, K., Gentili, R., Reggia, J., Gupta, S.: Incorporating failure-to-success transitions in imitation learning for a dynamic pouring task. In: IROS Workshop on Compliant Manipulation. (2014)
6. Okada, K., Kojima, M., Sagawa, Y., Ichino, T., Sato, K., Inaba, M.: Vision based behavior verification system of humanoid robot for daily environment tasks. In: Humanoids. (2006) 7–12
7. Tamosiunaite, M., Nemec, B., Ude, A., Wörgötter, F.: Learning to pour with a robot arm combining goal and shape learning for dynamic movement primitives. Robotics and Autonomous Systems **59**(11) (2011) 910–922
8. Cakmak, M., Thomaz, A.L.: Designing robot learners that ask good questions. In: HRI, ACM (2012) 17–24
9. Rozo, L., Jimenez, P., Torras, C.: Force-based robot learning of pouring skills using parametric hidden markov models. In: RoMoCo. (2013) 227–232
10. Rankin, A., Matthies, L.: Daytime water detection based on color variation. In: IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). (2010) 215–221
11. Rankin, A.L., Matthies, L.H., Bellutta, P.: Daytime water detection based on sky reflections. In: IEEE International Conference on Robotics and Automation (ICRA). (2011) 5329–5336
12. Griffith, S., Sukhoy, V., Wegter, T., Stoytchev, A.: Object categorization in the sink: Learning behavior–grounded object categories with water. In: Proceedings of the 2012 ICRA Workshop on Semantic Perception, Mapping and Exploration, Citeseer (2012)
13. Long, J., Shelhamer, E., Darrell, T.: Fully convolutional networks for semantic segmentation. In: CVPR. (2015) 3431–3440
14. Havaei, M., Davy, A., Warde-Farley, D., Biard, A., Courville, A., Bengio, Y., Pal, C., Jodoin, P.M., Larochelle, H.: Brain tumor segmentation with deep neural networks. arXiv preprint arXiv:1505.03540 (2015)
15. Romera-Paredes, B., Torr, P.H.: Recurrent instance segmentation. arXiv preprint arXiv:1511.08250 (2015)
16. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural computation **9**(8) (1997) 1735–1780
17. Greff, K., Srivastava, R.K., Koutník, J., Steunebrink, B.R., Schmidhuber, J.: Lstm: A search space odyssey. arXiv preprint arXiv:1503.04069 (2015)
18. Oh, J., Guo, X., Lee, H., Lewis, R.L., Singh, S.: Action-conditional video prediction using deep networks in atari games. In Cortes, C., Lawrence, N.D., Lee, D.D., Sugiyama, M., Garnett, R., eds.: NIPS. (2015) 2863–2871
19. Blender Online Community: Blender - A 3D modelling and rendering package. Blender Foundation, Blender Institute, Amsterdam. (2016)
20. Körner, C., Pohl, T., Rüde, U., Thürey, N., Zeiser, T.: Parallel lattice boltzmann methods for cfd applications. In: Numerical Solution of Partial Differential Equations on Parallel Computers. Springer (2006) 439–466
21. Jia, Y., Shelhamer, E., Donahue, J., Karayev, S., Long, J., Girshick, R., Guadarrama, S., Darrell, T.: Caffe: Convolutional architecture for fast feature embedding. arXiv preprint arXiv:1408.5093 (2014)
22. Kingma, D., Ba, J.: Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980 (2014)
23. Levine, S., Koltun, V.: Guided policy search. In: ICML (3). (2013) 1–9