# Use the Force, Luke!
# Learning to Predict Physical Forces by Simulating Effects
# Supplementary Materials

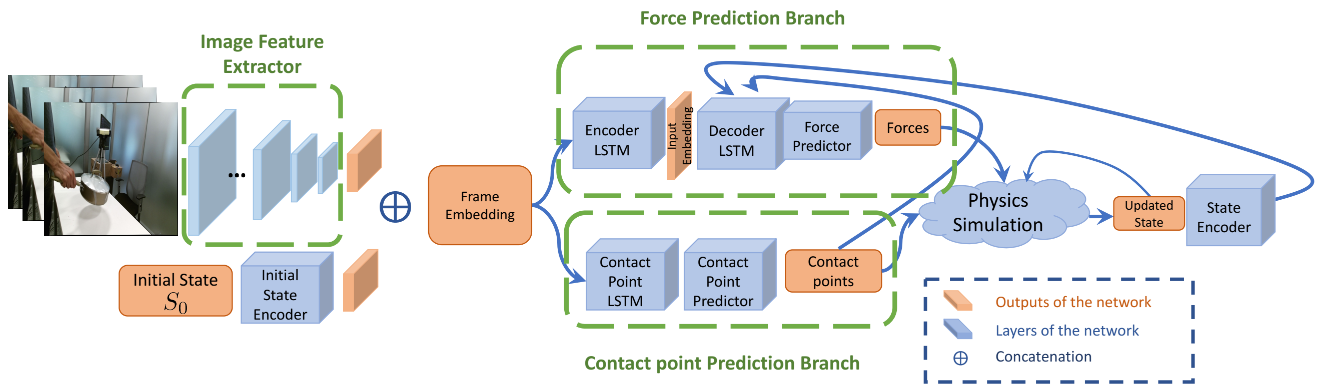


**Figure 1: Architecture details.**

## 1. Supplementary Material

We first explain the details of the data collection, and data pre-processing. Then we continue by discussing the architecture and hyper-parameters in detail to navigate the researchers to reproduce the results. Code in PyTorch and the dataset will be released to the public later. Finally, we illustrate one frame per object showing the simulated trajectory, the contact points and the forces applied to them at a specific time step (Figure 2). Code, data and the video of qualitative results are available in `https://ehsanik.github.io/forcecvpr2020`.

### 1.1. Dataset

**Annotations for Motion Estimation.** We manually defined 10 keypoints per object and asked Turkers to annotate these points on each video frame if the corresponding keypoint was visible. Given the known locations of these points of the object mesh, and their (annotated) projections on the image, we can recover the object's 6D pose w.r.t. the camera using the PnP algorithm. However, we note that due to noisy annotations, partial visibility, or co-planarity of points, the recovered poses are often inaccurate.

**Annotating Contact Points.** Instead of directly marking contact points for each interaction on the object mesh, we note that finding their projections on the image *i.e.* pixel locations of fingers is more natural for annotators. Given the annotations of their projections across the interaction frames, we can find the contact points on the mesh surface that project at these locations under the poses $(R_t, T_t)$ inferred in the previous step. This is achieved by solving an optimization problem of finding $C$ for each interaction sequence such that,

$$arg\,min_C \sum_t \|\pi(C, R_t, T_t) - l_t^c\|^2, \qquad (1)$$

where $t = 0, \ldots, n$ is the time step, $C \in \mathbb{R}^{k\times 3}$ is the set of contact points (k is the number of contact points), $R_t$ and $T_t$ are the rotation and translation of the object in world coordinate, respectively, $l_t^c$ is the pixel annotation for contact points from Mechanical Turk, and $\pi$ is the projection of the 3D points from the world coordinate to the camera frame. Note that the camera is static and does not move throughout each interaction sequence.
Detailed statistics of the dataset is available in Table 1.

### 1.2. Architecture Details

The initial state encoder, contact point predictor and force predictor in Figure 1 are each three fully connected layers. Image feature extractor is a Resnet18 without the last fully connected and average pooling layer followed by a pointwise convolution. Encoder LSTM and contact point LSTM are each a unidirectional Long-Short Term Memory with 3 layers of hidden size 512 and Decoder LSTM is a single LSTM cell with hidden size 512. The inputs to Decoder LSTM at each time step $t$, are the encoding for the current state $S_t$, the predicted contact points $C_t$, the input embedding (which is the embedding of the entire sequence), and

| Object | Train | Test | Validation |
|---|---|---|---|
| Pitcher | 843/12 | 135/4 | 219/3 |
| Bleach | 994/15 | 298/5 | 217/3 |
| Skillet | 904/12 | 336/4 | 397/4 |
| Drill | 1202/14 | 351/4 | 344/5 |
| Hammer | 970/13 | 297/4 | 302/4 |
| Plane | 1762/19 | 444/6 | 339/6 |
| Tomato soup | 634/12 | 224/3 | 352/3 |
| Mustard | 1007/14 | 299/5 | 384/4 |
| Total | 8316/111 | 2384/31 | 2554/32 |

**Table 1: Dataset Statistics** Distribution of number of frames and number of distinct contact points per split. (#frames/# distinct contact points)

the frame embedding for time $t + 1$. State encoder is three fully connected layers that embed current rotation and translation of the object as well as its linear and angular velocity.

### 1.3. Training Details

To avoid over-fitting and enhance generalization, we randomly jitter the hue, saturation, and brightness of the images by $0.05$. We train each one of our models until convergence on the training set, which takes between 30-60 epochs. Training each epoch takes 18-35 minutes on one GPU and 12 core CPU. We use a learning rate of $0.0001$ for few shot experiments and $0.001$ for the rest. We resize the input images to $224 \times 224$ before giving them as input to the feature extraction block.

**Qualitative results follow on the next page.**

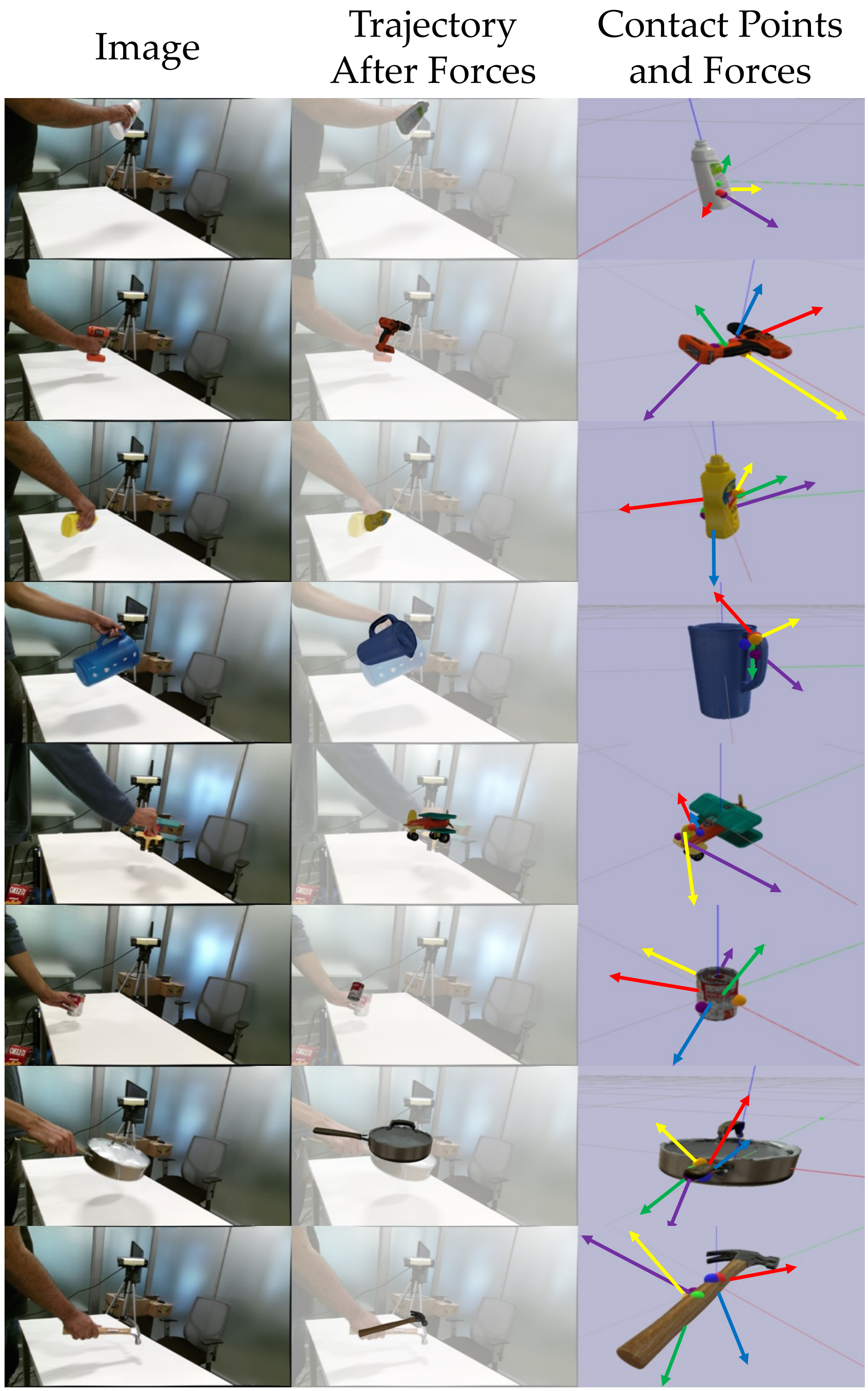


**Figure 2: More qualitative frames.** We show the estimated contact points and forces on variety of objects. For more videos and contact point visualizations refer to project's webpage (https://ehsanik.github.io/forcecvpr2020).