"What happens if..." Learning to Predict the Effect of Forces in Images

Roozbeh Mottaghi[1], Mohammad Rastegari[1], Abhinav Gupta[1,2], Ali Farhadi[1,3]

[1]Allen Institute for AI, [2]Carnegie Mellon University, [3]University of Washington

Abstract. *What happens if* one pushes a cup sitting on a table toward the edge of the table? How about pushing a desk against a wall? In this paper, we study the problem of understanding the movements of objects as a result of applying external forces to them. For a given force vector applied to a specific location in an image, our goal is to predict long-term sequential movements caused by that force. Doing so entails reasoning about scene geometry, objects, their attributes, and the physical rules that govern the movements of objects. We design a deep neural network model that learns long-term sequential dependencies of object movements while taking into account the geometry and appearance of the scene by combining Convolutional and Recurrent Neural Networks. Training our model requires a large-scale dataset of object movements caused by external forces. To build a dataset of forces in scenes, we reconstructed all images in SUN RGB-D dataset in a physics simulator to estimate the physical movements of objects caused by external forces applied to them. Our Forces in Scenes (ForScene) dataset contains 65,000 object movements in 3D which represent a variety of external forces applied to different types of objects. Our experimental evaluations show that the challenging task of predicting long-term movements of objects as their reaction to external forces is possible from a single image. The code and dataset are available at: http://allenai.org/plato/forces.

Keywords: Scene Understanding, Forces, Motion Estimation, Recurrent Neural Networks

1 Introduction

An important component in visual reasoning is the ability to understand the interaction between forces and objects; and the ability to predict the movements caused by those forces. We humans have an amazing understanding of how applied and action-reaction forces work. In fact, even with a static image [10,1], humans can perform a mental simulation of the future states and reliably predict the dynamics of the interactions. For example, a person can easily predict that the couch in Figure 2(a) will not move if it is pushed against the wall and the mouse in Figure 2(b) will eventually drop if it is pushed towards the edge of a desk.

In this paper, we address the problem of predicting the effects of external forces applied to an object in an image. Figure 1 shows a long-term prediction of the sequence of movements of a cup when it is pushed toward the edge of the table. Solving this problem requires reliable estimates of the scene geometry, the underlying physics, and

Fig. 1. Our goal is to learn "What happens if Force X is applied to Point Y in the scene?". For example, from a single image, we can infer that the cup will drop if we push it towards the edge of the table. On the right we show the output of our method, i.e. a sequence of velocities in 3D.

the semantic and geometric properties of objects. Additionally, it requires reasoning about interactions between forces and objects where subtle changes in how the force is applied might cause significant differences in how objects move. For example, depending on the magnitude of the force, the cup remains on the table or falls. What makes this problem more challenging is the sequential nature of the output where predictions about movements of objects depend on the estimates from the previous time steps. Finally, a data-driven approach to this problem requires a large-scale training dataset that includes movements of objects as their reaction to external forces. Active interaction with different types of scenes and objects to obtain such data is non-trivial.

Most visual scene understanding methods (e.g., [25,9,40]) are *passive* in that they are focused on predicting the scene structure, the objects, and their attributes and relations. These methods cannot estimate what happens if some parts of the scene are changed actively. For example, they can predict the location or 3D pose of a sofa, but they cannot predict how the sofa will move if it is pushed from behind. In this paper, we focus on an *active* setting, where the goal is to predict *"What happens if Force X is applied to Point Y in the scene?"*.

We design a deep neural network model that learns long-term sequential dependencies of object movements while taking into account the geometry and appearance of the scene by combining Convolutional and Recurrent Neural Networks. The RNN learns the underlying physical rules of movements while the CNN implicitly encodes the appearance and geometry of the object and the scene. To obtain a large number of observations of forces and objects to train this model, we collect a new dataset using physics engines; current datasets in the literature represent static scenes and are not suitable for active settings. Instead of training our model on synthetic images we do the inverse: we replicate the scenes in SUN RGB-D dataset [32] in a physics engine. The physics engine can then simulate forward the effect of applying forces to different objects in each image. We use the original RGB images, the forces, and their associated movements to form our dataset for training and evaluation.

Our experimental evaluations show that the challenging task of predicting long-term movements of objects as their reaction to external forces is possible from a single image. Our model obtains promising results in predicting the direction of the velocity of objects in 3D as the result of applying forces to them. We provide results for different variations of our method and show that our model outperforms baseline methods that perform

Fig. 2. Subtle differences in forces cause significantly different movements. The force is shown in yellow and the direction of movement is shown in green. (a) No movement is caused by the force since there is a wall behind the sofa. (b) The force changes the height of the object. The mouse drops as the result of applying the force. (c) The object might move in the opposite direction of the force. The chair initially moves in the direction of the force, but it bounces back when it hits the desk. (d) The direction of the movement and the force is the same.

regression and nearest neighbor search using CNN features. Furthermore, we show that our method generalizes to object categories that it has not seen during training.

2 Related Work

Passive scene understanding. There is a considerable body of work on scene understanding in the vision literature, e.g., [25,9,40,20,12,31,4,21,42]. However, most of these works propose *passive* approaches, where they infer the *current* configuration of the scenes (location of the objects, their 3D pose, support relationship, etc.) depicted in images or videos. In contrast, our method is an *active* approach, where we predict the result of interacting with objects using forces.

Physics-based prediction. [24] estimate the forces applied to the object and its trajectory for an action that is already happening in a scene. In this paper, we predict the future movements of the object for a *given force*. [8] predict the effect of forces in a billiard scene. Our method infers the movements based on a single image, while [8] uses a sequence of images. Also, [8] works on synthetic billiard scenes, while our method works on realistic images. [44] detect potentially falling objects given a point cloud representing the scene. In contrast, our method is based solely on visual cues and does not explicitly use physics equations.

Estimating physical properties. [2] estimate the physical parameters of rigid objects using video data. [3] estimates forces applied to a human using the dynamics of contacts with different surfaces. [39] learn a model for estimating physical properties of objects such as mass and friction based on a series of videos that show movement of objects on an inclined surface. These methods are not designed to predict the result of applying new forces to the scene and are limited to their controlled settings.

Stability inference. [43] reasons about the stability of objects in a given point cloud. [13] solves a joint optimization for segmentation, support relationships and stability. [14] propose a method to place a new object in a stable and semantically preferred location in a scene. Our method, in contrast, predicts the future movements of objects caused by applying forces to them.

Predicting sequences using neural networks. [28] propose a recurrent architecture to predict future frames of a video. [26] propose a recurrent neural net to predict the next frame in an Atari game given the current action and the previous frames. [33] propose Recurrent RBMs to model high dimensional sequences. [23] model temporal dependencies of a sequence and predict multiple steps in the future. These approaches either require a full sequence (past states and current actions) or work only on synthetic data and in limited environments. Also, [19] propose a deep-learning based method to perform a pre-defined set of tasks. They learn a distribution over actions given the current observation and configurations. In contrast, we predict how the scene changes as the result of an action (i.e. applying forces). In the language domain, [15,34] have used a combination of CNNs and RNNs to generate captions for images.

Data-driven prediction. [36] infers the future path of rigid objects according to learned models of appearance, context, and transition. [27,37] predict optical flow from a single image. [41] predict future events that might take place in a query image. [16] estimate future movements of humans in a given scene. [7] predicts relative movements of objects. Unlike these approaches, we explicitly represent forces and focus on the physics of the scene in order to infer future movements of objects in 3D.

Physics-based tracking. [30] recover 3D trajectories and the forces applied to objects in a tracking framework. [35] incorporates physical plausibility into a human tracking framework. Our problem is different from tracking since we perform inference using a single image.

3 Problem Statement

Given a query object in a single RGB image and a force vector, our goal is to predict the future movement of the object as the result of applying the force to the object. More specifically, for a force $\boldsymbol{f}$ and an impact point $\boldsymbol{p}$ on the object surface in the RGB image, our goal is to estimate a variable-length sequence of velocity directions $V = (v_0, v_1, \ldots, v_t)$ for the center of the mass of the object. These velocities specify how the location of the object changes over time.

For training we need to obtain the sequence V that is associated to force $\boldsymbol{f} = (f_x, f_y, f_z)$ applied to point $\boldsymbol{p} = (p_u, p_v)$ in the image. To this end, we automatically synthesize the scene in a physics engine (described in Section 4). The physics engine simulates forward the effect of applying the force to the point that corresponds to $\boldsymbol{p}$ in the 3D synthetic scene and generates the velocity profile and locations for the query object.

During testing, we do not have access to the synthesized scene or the physics engine, and our goal is to predict the sequence V given a query object in a single RGB image and a force[1].

We formulate the estimation of the movements as a sequential classification problem. Hence, each v_t takes a value from the set $\mathcal{L} = \{l_1, l_2, \ldots, l_N, s\}$, where each l_i denotes the index for a direction in the quantized space of 3D directions, and s represents 'stop' (no motion). The velocity at each time step v_t depends on the previous

[1] We refer to the force and its impact point as *force* throughout the paper.

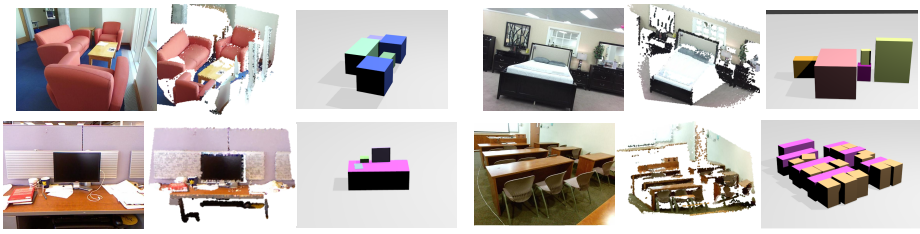

Fig. 3. Synthetic scenes. These example scenes are synthesized automatically from the images in the SUN RGB-D [32] dataset. Left: the original image, Middle: point cloud representation, Right: synthetic scene. The objects that belong to the same category are shown with the same color. For clarity, we do not visualize the walls.

movements of the object. Therefore, a natural choice for modeling these temporal dependencies is a recurrent architecture. To couple the movement information with the appearance and geometry of the scene and also the force representation, our model integrates a Recurrent Neural Network (RNN) with a Convolutional Neural Network (CNN). Section 5 describes the details of the architecture.

4 Forces in Scenes (ForScene) Dataset

One of the key requirements of our approach is an *interactable* dataset. Most of the current datasets in the vision community are 'static' datasets in that we cannot apply forces to objects depicted in the scenes and modify the scenes. For example, we cannot move the constituent objects of the scenes shown in PASCAL [6] or COCO [22] images as we desire since inferring the depth map and the physics of the scene from a single RGB image is a challenging problem. An alternative would be to use RGB-D images, where the depth information is available. This solves the problem of depth estimation and moving the objects in perspective, but RGB-D images do not provide any information about the physics of the world either.

To make an *interactable* dataset, we transfer the objects and the scene layout shown in images to a physics engine. The physics engine takes a scene and a force as input and simulates the future states of the objects in the scene according to the applied forces. This enables us to collect the velocity sequences that we require for training our model.

Our dataset is based on the SUN RGB-D dataset [32], which includes 2D and 3D annotations in the form of 2D semantic segmentation and 3D bounding boxes for about 1,000 object categories. The 3D position and orientation of each bounding box is provided in the annotations, hence, we can transfer the 3D bounding boxes of the objects to the physics engine and reconstruct the same object arrangement in the physics engine. In addition, the SUN RGB-D dataset includes annotations for the scene layout (floors, walls, etc). We replicate the scene layout in the physics engine as well. Figure 3 shows a few examples of the images and their corresponding scenes in the physics engine. More details about the dataset can be found in Section 6.1. Note that our training and evaluation is performed on real images. These synthetic scenes only supply the groundtruth velocity information.

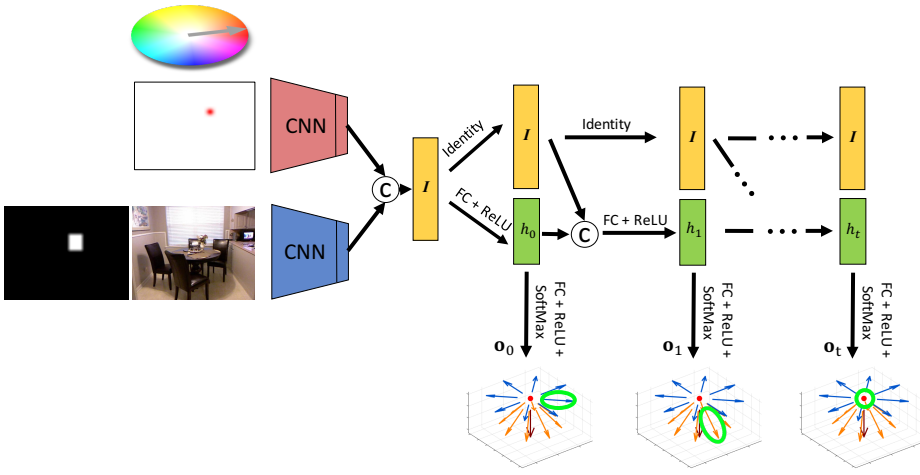

Fig. 4. Model. Our model consists of two CNNs for capturing the *force* and *image* information. We refer to these CNNs as force tower and image tower respectively. The input to the model is a force image and an RGB-M image (RGB image plus an M channel representing object bounding box). The color in the force image represents the direction and magnitude of the force (according to the color wheel). The symbol Ⓒ denotes concatenation. 'Identity' propagates the input to the output with no change. h_t represents the hidden layer of the RNN at time step t. Also, we use the abbreviation FC for a fully connected layer. The output of our model is a sequence of velocity *directions* at each time step. We consider 17 directions and an additional 'stop' class, which is shown by a red circle. The green ellipses show the chosen direction at each time step. The RNN stops when it generates the 'stop' class.

5 Model

We now describe different components of our model, how we represent objects and forces in the model and how we formulate the problem to predict the movements.

5.1 Model architecture

Our model has three main components: (1) A Convolutional Neural Network (CNN) to encode scene and object appearance and geometry. We refer to this part of the model as *image tower*. (2) Another CNN (parallel to the image tower) to capture force information. We refer to this part of the model as *force tower*. (3) A Recurrent Neural Network (RNN) that receives the output of the two CNNs and generates the object motion (or equivalently, a sequence of vectors that represent the velocity of the object at each time step). Note that the training is end-to-end and is performed jointly for all three components of the model. Figure 4 illustrates the architecture of the full network.

We use two different architectures for the image tower for the experiments: AlexNet [17] and ResNet-18 [11], where we remove their final classification layer. Similar to [24], the input to our image tower is a four-channel RGB-M image, where we add a mask channel (M) to the RGB image. The mask channel represents the location of

the query object and it is obtained by applying a Gaussian kernel to a binary image that shows the bounding box of the query object. We propagate the output of the layer before the last layer of the CNN (e.g., FC7 when we use AlexNet) to the next stages of the network.

The force tower is structured as an AlexNet [17] and is parallel to the image tower. The input to the force tower is an RGB image that represents the impact point, direction and magnitude of the query force (we will explain in Section 6.2 how this image is created). The output of the FC7 layer of the force tower is propagated to the next stages of the network. Our experiments showed that using a separate force tower provides better results compared to adding the force as another input channel to the image tower. Probably, the reason is that there is too much variability in the real images, and the network is not able to capture the information in the force image when we have a single tower for both real images and force images. Therefore, we consider two separate towers and combine the output of these towers at a later stage. The outputs of the image tower and force tower are concatenated (referred to as I in Figure 4) and provide a compact encoding of the visual cues and force representation for the recurrent part of the network.

The recurrent part of our network receives I as input and generates a sequence of velocity vectors. The advantage of using a Recurrent Neural Network (RNN) is twofold. First, the velocities at different time steps are dependent on each other, and the RNN can capture these temporal dependencies. Second, RNNs enable us to predict a variable-length sequence of velocities (the objects move different distances depending on the magnitude of the force and the structure of the scene). We show the unfolded RNN in Figure 4. The hidden layer of the RNN at time step t is a function of I and the previous hidden unit (h_{t-1}). More formally, $h_t = f(I, h_{t-1})$, where f is a linear function (fully connected layer) followed by a non-linear ReLU (Rectified Linear Unit). [18] show that RNNs composed of ReLUs and initialized with identity weight matrix are as powerful as standard LSTMs. The first hidden unit of the RNN (h_0) is only a function of I. The output at each time step $\mathbf{o}_t$ is a function of the hidden layer h_t. More concretely, $\mathbf{o}_t = \text{SoftMax}(g(h_t))$, where g is a linear function, which is augmented by a ReLU.

We use 1000 neurons for the hidden layer in the recurrent part of the network. The output $\mathbf{o}_t$ is of size $|\mathcal{L}|$. $\mathcal{L}$, as defined in Section 3, is a set of directions in 3D and a 'stop' class, which represents the end of the sequence.

5.2 Training

To train our model, in each iteration, we feed a random batch of RGB-M images from the training set into the image tower. The corresponding batch of force images is fed into the force tower. There is a sequence of velocity vectors associated to each pair of RGB-M and force images. These sequences have different lengths depending on the velocity profile of the query object in the groundtruth. If the object does not move as the result of applying the force, the sequence will be of length 1, where its value is 'stop'. The training is performed end-to-end, and each iteration involves a forward and a backward pass through the entire network.

The loss function is defined over the sequence of outputs $O = (\mathbf{o}_0, \mathbf{o}_1, \ldots, \mathbf{o}_{t'})$. Suppose the groundtruth velocity sequence is denoted by $V = (v_0, v_1, \ldots, v_t)$, the classification loss, $E(V, O)$, which is based on the cross entropy loss, is defined as follows:

$$E(V, O) = -\frac{1}{T} \sum_{t=0}^{T} q_t(v_t) \log(\mathbf{o}_t[v_t]), \tag{1}$$

where $\mathbf{o}_t[v_t]$ represents the v_t-th element of $\mathbf{o}_t$, T is the maximum length of a sequence, and $q_t(v_t)$ is the inverse frequency of direction v_t in step t of the sequences in the training data. We pad the end of the sequences whose length is shorter than T (i.e. $|O| < T$ or $|V| < T$) with 'stop' so their length becomes equal to T. We could alternatively represent velocities as 3-dimensional vectors and use a regression loss instead. However, we achieved better performance using the classification formulation. A similar observation has been made by [38,37] that formulate a continuous variable estimation problem as classification.

5.3 Testing

The procedure for predicting a sequence of velocity vectors is as follows. We obtain I (the input to the RNN) by feeding the RGB-M and force images into the object and force towers, respectively. The hidden unit h_0 is computed according to the fully connected layer that is defined over I. The first velocity in the sequence, v_0, is computed by taking the argmax of the output of the SoftMax layer that is defined over h_0. We compute h_1 based on I and h_0 and similarly find the next velocity, v_1, in the sequence. More concretely, $v_t = \arg\max \mathbf{o}_t$ (recall that v_t is the index for a direction in the quantized set of directions or 'stop'). We continue this process until the RNN generates the 'stop' class (i.e. $v_t =$ stop) or it reaches the maximum number of steps that we consider.

6 Experiments

In this section, we describe the evaluation of our method and compare our method with a set of baseline approaches. We provide the details of the dataset and explain how we interact with objects in the scenes. Additionally, we explain how we represent the force in the CNN and provide more implementation details about our network.

6.1 Dataset details

Our dataset is based on the SUN RGB-D [32] dataset, which contains 10,335 images (we ignore 1,548 images that miss wall annotations or are not fully annotated). Each object is annotated with a 3D bounding box (position and orientation in 3D) and a segmentation mask (a 2D segmentation mask in the RGB image). There are more than 1,000 object categories in the dataset. Additionally, the room layout annotations (in the form of walls and floors) are provided for each image in the dataset. These annotations enable us to automatically reconstruct a similar scene in a physics engine. Therefore,

for each image in [32], we have a synthetic scene, which will be used to simulate the effect of the forces.

We use Blender physics engine[2] to render the synthetic scenes. Some example scenes and their corresponding images are shown in Figure 3[3]. To create our dataset, we use all ∼1,000 categories and walls and floors to construct the synthetic scene, however, we apply the force to the 50 most frequent rigid categories in the dataset. These categories include: chair, keyboard, flower vase, etc. We represent each object as a cube in the synthetic scene.

For each object in the image, we randomly select a point on the surface of the object and apply the force to this point (note that during training for each point in the RGB image, we know the corresponding 3D point in the synthetic scene). The input force is also chosen at random. We simulate the scene after applying the force to the impact point. The simulation continues until the object to which the force is applied reaches a stable state, i.e. the linear and angular velocities of the object become zero. Over the entire dataset, it took a maximum of 32 simulation steps that the object converges to the stable position. We sample velocities every 6 steps, which results in a sequence of at most 6 velocity vectors (depending on the number of steps needed for reaching stability). We use this sequence as the groundtruth sequence for the query object and force. We represent these velocities in a quantized space of 3D directions (we ignore the magnitude of the velocities), where the directions are 45 degrees apart from each other. Figure 4 shows these directions. We have 17 directions in total, hence, the size of the set $\mathcal{L}$ (defined in Section 3) will be 18 (17 directions + 1 'stop' class). We assign the velocity vector to the nearest direction class using angular distance. If the magnitude of the velocity vector is lower than a threshold we assign it to the 'stop' class. These directions cover a semi-sphere since the velocity directions in the other semi-sphere are rare in our dataset.

As the result of the simulations, we obtain 30,655 velocity sequences for training and validation and 34,777 sequences for test. Note that sometimes we apply the force in the same direction but with different magnitudes. In the real world, some of the objects such as toilets or kitchen cabinets are fixed to the floor. We consider those object categories as 'static' in the physics engine, which means we cannot move them by applying a force. Figure 5 shows a sequence of movement in a synthetic scene.

6.2 Force representation

To feed the force to the CNN, we convert the force vector to an RGB image. Here we describe the procedure for creating the force image. For simplicity, we set the z component of our forces to zero (we refer to the axis that is perpendicular to the ground as the z axis). However, note that the z component of their corresponding velocities can be non-zero (e.g., a falling motion). The force image is the same size as the input RGB image. We represent the force as a Gaussian that is centered at the impact point of the force in the 2D image. We use a color from a color wheel (shown in Figure 4) to

[2] http://www.blender.org

[3] Some objects are missing in the synthetic scenes since they are not annotated in the SUN RGB-D dataset.

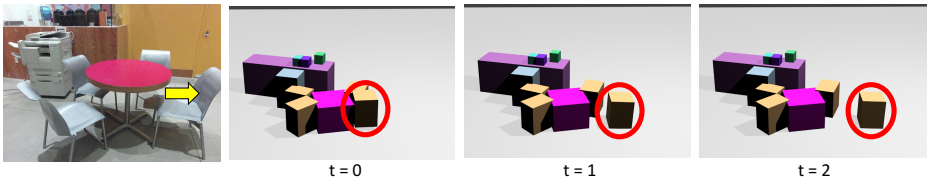

Fig. 5. Synthesizing the effect of the force. A force (shown by a yellow arrow) is applied to a point on the surface of the chair. The three pictures on the right show different time steps of the scene simulated in the physics engine. There is a red circle around the object that moves.

represent the direction and the magnitude of the force. Each point on the color wheel specifies a unique direction and magnitude. The standard deviation of the Gaussian is 5 pixels in both directions.

6.3 Network and optimization parameters

We used Torch[4] to implement the proposed neural network. We run the experiments on a Tesla K40 GPU. We feed the training images to the network in batches of size 128 when we use AlexNet for the image tower and of size 96 when we use ResNet-18 for the image tower. Our learning rate starts from 10^{-2} and gradually decreases to 10^{-4}. We initialize the image tower and the force tower by a publicly available AlexNet model[5] or ResNet model[6] that are pre-trained on ImageNet. We randomly initialize the 4th channel of the RGB-M image (the M channel) by a Gaussian distribution with mean 0 and standard deviation 0.01. The forward pass and the backward pass are performed for 15,000 iterations when we use AlexNet for the image tower (the loss value does not change after 15K iterations). When we use ResNet-18 we use 35,000 iterations since it takes longer to converge.

6.4 Prediction of velocity sequences

We evaluate the performance of our method on predicting the 34,777 sequences of velocity vectors in the test portion of the dataset.

Evaluation criteria. To evaluate the performance of our method, we compare the estimated sequence of directions with the groundtruth sequence. If the predicted sequence has a different length compared to the groundtruth sequence, we consider it as incorrect. If both sequences have the same length, but they differ in at least one step, we consider that as an incorrect prediction as well. We report the percentage of sequences that we have predicted entirely correctly. We have about 1000 patterns of sequences in our test data so the chance performance is close to 0.001.

Results. We estimate 16.5% of the sequences in the test data correctly using our method that uses AlexNet as image and force towers. We refer to this method as 'ours w/

[4] http://torch.ch

[5] http://github.com/BVLC/caffe/tree/master/models/bvlc_alexnet

[6] https://github.com/facebook/fb.resnet.torch/tree/master/pretrained

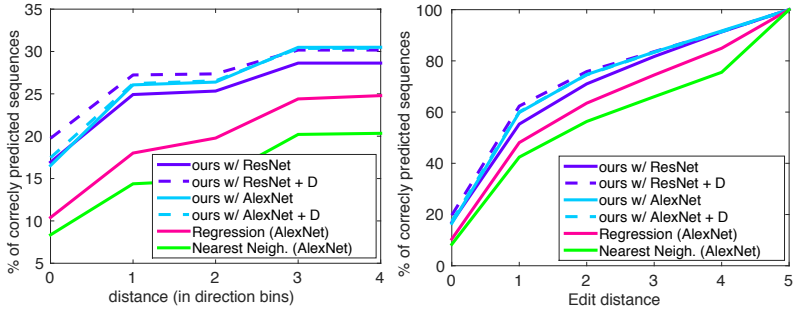

Fig. 6. Relaxation of the evaluation criteria. (a) We consider the prediction for each step as correct if it is among the k nearest directions to the groundtruth direction. The x-axis shows k. (b) We consider a predicted sequence as correct if it is within edit distance k of the groundtruth sequence. The x-axis shows k.

AlexNet' in Table 1. The criteria that we consider is a very strict criteria. Therefore, we also report our results using less strict criteria. We consider a direction as correct if it is among the closest k directions to the groundtruth direction. Figure 6(a) shows these results for $k = 0, \ldots, 4$ ($k = 0$ means we compare with the actual groundtruth class). We observe a significant improvement using this relaxed criteria. We also report the results using 'edit distance', which is a measure of dissimilarity between the groundtruth and the predicted sequences. Basically, it measures how many operations we need to convert a sequence to the other sequence. We report what percentage of predicted sequences are correct within edit distances 0 to 5. This result is shown in Figure 6(b). The result of 'ours w/ AlexNet' improves to 59.8% from 16.5% if we consider the predictions whose edit distance with the groundtruth is less than or equal to 1, as correct.

We also replaced the AlexNet in the image tower by the ResNet-18 [11] model. The performance for this case (referred to as 'ours w/ ResNet') is reported in Table 1. The results using the relaxed criteria are shown in Figures 6(a) and 6(b). To analyze the effect of depth on the predictions, we also incorporated depth into the image tower. We add the depth image as another channel in the input layer of the image tower. For obtaining the depth images, we use the method of [5], which estimates depth from a single image. We use their publicly available model, which is trained on a subset of the SUN RGB-D dataset. Using depth improves 'ours w/ ResNet' and 'ours w/ AlexNet' by 2.9% and 1.0%, respectively (Table 1). It seems ResNet better leverages this additional source of information. We initialize the additional depth channel randomly (random samples from a Gaussian distribution with mean 0 and standard deviation 0.01). The results for these ablative cases using the relaxed criteria are also shown in Figure 6.

Some qualitative results are shown in Figure 7. For example, Figures 7(a) and (c) show two cases that the object moves in the same direction as the force. Figure 7(b) shows an example of falling, where the lamp moves straight for two steps and then it drops. Figure 7(e) shows an example that the object bounces back as the result of

ours w/ ResNet + Depth	ours w/ ResNet	ours w/ AlexNet + Depth	ours w/ AlexNet	Regression AlexNet	Nearest Neigh. AlexNet
19.8	16.9	17.5	16.5	10.4	8.5

Table 1. Ablative analysis of our method and comparison with baseline approaches. The evaluation metric is the percentage of sequences that we predict correctly.

chair	table	desk	pillow	sofa chair	sofa	bed	box	garbage bin	shelf	**Avg.**	**All**
17.7	17.0	15.4	15.5	15.9	17.2	15.9	14.6	16.1	15.9	16.12	16.53

Table 2. Generalization of the method to the classes that were not seen during training. Each column shows the results for the case that we remove the sequences corresponding to that category from the training set. The rightmost column ('All') shows the base case, where all training examples are seen.

applying a large force. Figure 7(f) shows an example that object does not move no matter how large the force is. It probably learns that pushing objects against a wall cannot cause a movement. There are two other examples in Figures 7(g) and (h), where the object does not move. We also show some failure cases in Figure 8. In Figure 8(a), the method ignores the wall behind the printer and infers a falling motion for the printer. In Figure 8(b) the stove goes through the cabinet, which is not a correct prediction. Note that the synthetic scenes are just for visualization of the movements and they are not used during testing and inference.

Baseline methods. The first baseline that we consider is a regression baseline, where we replace the RNN part of our network with a regressor that maps I (refer to Figure 4) to 18 numbers (we have at most 6 steps and at each step we want to predict a 3-dimensional vector). If the length of the training sequence is less than 6, we set their corresponding elements in the 18-dimensional vector to zero. We use a smooth L1 loss function. As the result of regression, we obtain a vector of size 18, which corresponds to six 3-dimensional vectors. We assign them to different bins in the quantized direction space or the 'stop' class (using the procedure described in Section 6.1). The results are reported in Table 1 and Figure 6. The result of the AlexNet-based regression method is 6.1% lower than the result of 'ours w/ AlexNet'.

Another baseline that we tried is a nearest neighbor baseline. For each query object and force in the test set, we forward the corresponding RGB-M and the force image to the our full network (which is already trained using our data). We obtain the features I. Then, we find the query object and force in our training data that produces the most similar I. We use the sequence that is associated to the most similar training data as the predicted sequence. The features are high dimensional. Hence, to find the nearest neighbor we use multiple index hashing method of [29]. The results of this AlexNet-based nearest neighbor is not competitive either (Table 1 and Figure 6).

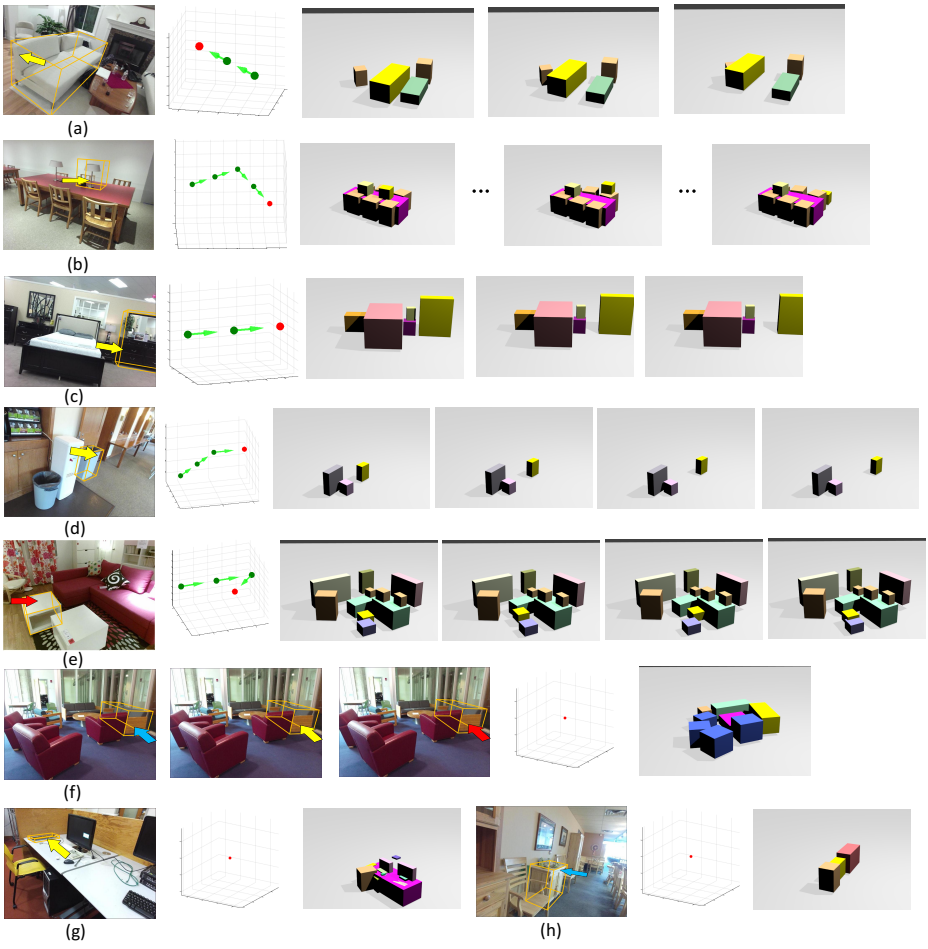

Fig. 7. Qualitative results. The left figure shows the force (color arrow) applied to the image. Different force magnitudes are shown with different colors, where blue, yellow, and red represent small, medium and large forces, respectively. The second image from the left shows the output of our method, which is a sequence of velocity vectors in 3D. The red point is the step that the velocity becomes zero. The resulted motion is visualized in the synthetic scenes. The object that moves is shown in yellow. Note that these synthetic scenes are for visualization purposes and they are not used during test. For clarity, we do not show walls.

6.5 Unseen categories

To evaluate how well our method generalizes to object categories that are not seen during training, we remove the training sequences that correspond to an object category and evaluate the method on the entire test set. For this experiment, we consider the ten most frequent object categories in our dataset.

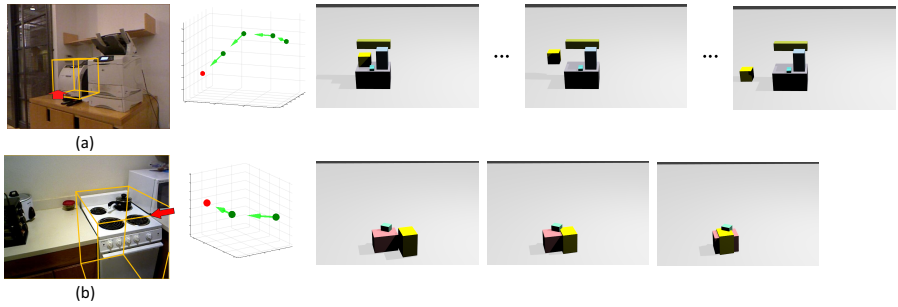

Fig. 8. Failure cases. For the details of the visualization, refer to the caption of Figure 7.

We re-train the network each time we remove the sequences corresponding to an object category from our training set. The result of this experiment is shown in Table 2. We report the results using the strict evaluation criteria. We use the method that we refer to as 'ours w/ AlexNet' for this experiment since its training time is faster than our other approaches. The results show that the average performance does not drop significantly compared to the case that we use the entire training set. This means that our method generalizes well to the categories that it has not seen during training.

7 Conclusion

Visual reasoning is a key component of any intelligent agent that is supposed to operate in the visual world. An important component in visual reasoning is the ability to predict the expected outcome of an action. This capability enables planing, reasoning about actions, and eventually successfully executing tasks. In this paper, we take one step toward this crucial component and study the problem of prediction the effect of an action (represented as a force vector) when applied to an object in an image. Our experimental evaluations show that our model can, in fact, predict sequential movements of objects when a force is applied to them.

Velocity magnitude and rotations: Our solution is mainly concerned with predicting translation vectors and velocity direction. Predicting velocity magnitude requires more complex reasoning about friction, materials, etc. Also our current model assumes uniform weights for all objects, resulting in a calibration issue for the magnitude of the force necessary to move an object. Our future work will involve addressing these issues.

Acknowledgements

This work is in part supported by ONR N00014-13-1-0720, ONR MURI N000141612007, NSF IIS- 1338054, Allen Distinguished Investigator Award, and the Allen Institute for Artificial Intelligence.

References

1. Battaglia, P., Hamrick, J., Tenenbaum., J.B.: Simulation as an engine of physical scene understanding. PNAS (2013) 1
2. Bhat, K.S., Seitz, S.M., Popovic, J.: Computing the physical parameters of rigid-body motion from video. In: ECCV (2002) 3
3. Brubaker, M.A., Sigal, L., Fleet, D.J.: Estimating contact dynamics. In: ICCV (2009) 3
4. Choi, W., Chao, Y.W., Pantofaru, C., Savarese, S.: Understanding indoor scenes using 3d geometric phrases. In: CVPR (2013) 3
5. Eigen, D., Puhrsch, C., Fergus, R.: Depth map prediction from a single image using a multi-scale deep network. In: NIPS (2014) 11
6. Everingham, M., Gool, L., Williams, C.K., Winn, J., Zisserman, A.: The pascal visual object classes (voc) challenge. IJCV (2010) 5
7. Fouhey, D.F., Zitnick, C.: Predicting object dynamics in scenes. In: CVPR (2014) 4
8. Fragkiadaki, K., Agrawal, P., Levine, S., Malik, J.: Learning predictive visual models of physics for playing billiards. In: ICLR (2016) 3
9. Gupta, A., Efros, A.A., Hebert, M.: Blocks world revisited: Image understanding using qualitative geometry and mechanics. In: ECCV (2010) 2, 3
10. Hamrick, J., Battaglia, P., Tenenbaum., J.B.: Internal physics models guide probabilistic judgments about object dynamics. Annual Meeting of the Cognitive Science Society (2011) 1
11. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: CVPR (2016) 6, 11
12. Heitz, G., Gould, S., Saxena, A., Koller, D.: Cascaded classification models: Combining models for holistic scene understanding. In: NIPS (2008) 3
13. Jia, Z., Gallagher, A., Saxena, A., Chen, T.: 3d-based reasoning with blocks, support, and stability. In: CVPR (2013) 3
14. Jiang, Y., Lim, M., Zheng, C., Saxena, A.: Learning to place new objects in a scene. IJRR (2012) 3
15. Karpathy, A., Fei-Fei, L.: Deep visual-semantic alignments for generating image descriptions. In: CVPR (2015) 4
16. Kitani, K.M., Ziebart, B.D., Bagnell, J.A.D., Hebert, M.: Activity forecasting. In: ECCV (2012) 4
17. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. In: NIPS (2012) 6, 7
18. Le, Q.V., Jaitly, N., Hinton, G.E.: A simple way to initialize recurrent networks of rectified linear units. In: ArXiv (2015) 7
19. Levine, S., Finn, C., Darrell, T., Abbeel, P.: End-to-end training of deep visuomotor policies. In: arXiv (2015) 4
20. Li, L.J., Socher, R., Fei-Fei, L.: Towards total scene understanding: Classification, annotation and segmentation in an automatic framework. In: CVPR (2009) 3
21. Lin, D., Fidler, S., Urtasun, R.: Holistic scene understanding for 3d object detection with rgbd cameras. In: ICCV (2013) 3
22. Lin, T.Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollr, P., Zitnick, C.L.: Microsoft coco: Common objects in context. In: ECCV (2014) 5
23. Michalski, V., Memisevic, R., Konda, K.: Modeling deep temporal dependencies with recurrent grammar cells. In: NIPS (2014) 4
24. Mottaghi, R., Bagherinezhad, H., Rastegari, M., Farhadi, A.: Newtonian image understanding: Unfolding the dynamics of objects in static images. In: CVPR (2016) 3, 6

25. Murphy, K., Torralba, A., Freeman, W.T.: Using the forest to see the trees: A graphical model relating features, objects, and scenes. In: NIPS (2003) 2, 3
26. Oh, J., Guo, X., Lee, H., Lewis, R.L., Singh, S.P.: Action-conditional video prediction using deep networks in atari games. In: NIPS (2015) 4
27. Pintea, S.L., van Gemert, J.C., Smeulders, A.W.M.: Déjà vu: - motion prediction in static images. In: ECCV (2014) 4
28. Ranzato, M., Szlam, A., Bruna, J., Mathieu, M., Collobert, R., Chopra, S.: Video (language) modeling: a baseline for generative models of natural videos. In: arXiv (2014) 4
29. Rastegari, M., Keskin, C., Kohli, P., Izadi, S.: Computationally bounded retrieval. In: CVPR (2015) 12
30. Salzmann, M., Urtasun, R.: Physically-based motion models for 3d tracking: A convex formulation. In: ICCV (2011) 4
31. Silberman, N., Hoiem, D., Kohli, P., Fergus, R.: Indoor segmentation and support inference from rgbd images. In: ECCV (2012) 3
32. Song, S., Lichtenberg, S.P., Xiao, J.: Sun rgb-d: A rgb-d scene understanding benchmark suite. In: CVPR (2015) 2, 5, 8, 9
33. Sutskever, I., Hinton, G.E., Taylor, G.W.: The recurrent temporal restricted boltzmann machine. In: NIPS (2008) 4
34. Vinyals, O., Toshev, A., Bengio, S., Erhan, D.: Show and tell: A neural image caption generator. In: CVPR (2015) 4
35. Vondrak, M., Sigal, L., Jenkins, O.C.: Physical simulation for probabilistic motion tracking. In: CVPR (2008) 4
36. Walker, J., Gupta, A., Hebert, M.: Patch to the future: Unsupervised visual prediction. In: CVPR (2014) 4
37. Walker, J., Gupta, A., Hebert, M.: Dense optical flow prediction from a static image. In: ICCV (2015) 4, 8
38. Wang, X., Fouhey, D.F., Gupta, A.: Designing deep networks for surface normal estimation. In: CVPR (2015) 8
39. Wu, J., Yildirim, I., Lim, J.J., Freeman, W.T., Tenenbaum, J.B.: Galileo: Perceiving physical object properties by integrating a physics engine with deep learning. In: NIPS (2015) 3
40. Yao, J., Fidler, S., Urtasun, R.: Describing the scene as a whole: Joint object detection, scene classification and semantic segmentation. In: CVPR (2012) 2, 3
41. Yuen, J., Torralba, A.: A data-driven approach for event prediction. In: ECCV (2010) 4
42. Zhang, Y., Song, S., Tan, P., Xiao, J.: Panocontext: A whole-room 3d context model for panoramic scene understanding. In: ECCV (2014) 3
43. Zheng, B., Zhao, Y., Yu, J.C., Ikeuchi, K., Zhu, S.C.: Beyond point clouds: Scene understanding by reasoning geometry and physics. In: CVPR (2013) 3
44. Zheng, B., Zhao, Y., Yu, J.C., Ikeuchi, K., Zhu, S.C.: Detecting potential falling objects by inferring human action and natural disturbance. In: ICRA (2014) 3