Reinforcement Learning with Deep Sensory Prediction Models for diverse Real-World Environments

TBD

Abstract—Applying deep reinforcement learning algorithms to robotic control tasks in complex, diverse open-world environments without access to precisely hand-engineered rewards is an open problem. We take a step towards solving this problem by drawing supervision from predicting future sensory observations. Concretely we propose to learn predictive models of future observations conditioned on the agent's future actions, and a method for planning actions under this model. We show that we can learn sensory prediction models effectively from real videos of a robot's experience, demonstrating that such models can handle diverse datasets with hundreds of objects, and that they enable the robot to accomplish a wide range of user-defined object manipulation goals, even when faced with novel objects. TO-DO: Page limit is 12!!

Index Terms—Deep Reinforcement Learning, Sensory Prediction Models, Video Prediction, Robotic Manipulation, Model Predictive Control

1 Introduction

Remarkably, humans are able to learn a range of complex yet generalizable behaviors when faced with a stream of high-dimensional sensory inputs and minimal external supervision.

The reinforcement learning (RL) problem statement covers such a real-world setting. Yet, the general setting of learning from raw sensory inputs from sparse rewards in the real world is incredibly challenging.

To make progress, many prior works have chosen various assumptions to make the problem more tractable. For instance, some works have been demonstrated only in settings where shaped rewards are available [**?**]. Others have assumed access to a perfect or hand-engineered simulator, from which millions of roll-outs can be taken [**?**], [**?**]. A few works have assumed that a low-dimensional, compact state representation is accessible during training [**?**], [1]. Finally, many works reset the state between roll-outs to a relatively narrow distribution of initial states, leading to a task is more repeatable [**?**], [**?**], [**?**], [2]. By making such assumptions, these prior methods have demonstrated the ability to learn complex visuomotor skills [1], learn Atari games from pixels [**?**], and master the game of Go [**?**]. However, to move these algorithms towards achieving general behaviors in the real world, it is arguably necessary to deploy them in diverse, real-world environments. Unfortunately, it is challenging to deploy such methods for learning in diverse, real-world environments, where shaped rewards, resets, compact state representations, and simulations are largely unavailable without extensive human involvement.

Unlike many prior works that focus on RL for mastery of highly-specialized skills in closed-world environments, here we instead focus on *generalization* in diverse environments. We consider a robotic control domain with a wide range of objects, where the algorithm has access to only raw sensor observations (i.e. pixels) and sparse rewards and has minimal control over the environment and how it can be reset. In such a setting with minimal assumptions, rewards are simply not enough supervision to be learned from alone. Instead, the supervision needs to come from the observations themselves. As such, we propose to learn action-conditioned predictive models directly on raw pixel observations. Beyond the necessity of dense supervision, sensory prediciton models have two additional benefits. They are goal-agnostic, which enables learning for a variety of different goals; and, by learning on top of raw observations, they are fully general in that they do not require a hand-engineered representation of the environment.

TO-DO: reference figure 1 here!

Despite the benefits, a number of challenges arise when aiming to use sensory prediction models: how can we learn a model of high-dimensional observations? how should an objective or reward be determined with respect to the predicted observations? how should experience be collected? how should actions be chosen with respect to the model? We discuss each of these design decisions and propose several options, weighting the benefits and trade-offs of each. Our overall approach amounts to a deep model-based reinforcement learning algorithm that leverages video prediction models to achieve a variety of pixel-based control tasks without shaped reward information.

The main contributions of this work are as follows. First, we propose a general framework for deep reinforcement learning with sensory prediciton models, abstracting away design decisions such as the model, the policy optimization, and the reward function. Second, we propose a model that can better maintain object permanence through occlusions via temporal skip connections. Third, we discuss several forms of planning objectives, including goal pixel positions,

- *Frederik Ebert*, Chelsea Finn*, Annie Xie, Sudeep Dasari, Alex Lee, Sergey Levine*

Manuscript received xx.xx.xxx

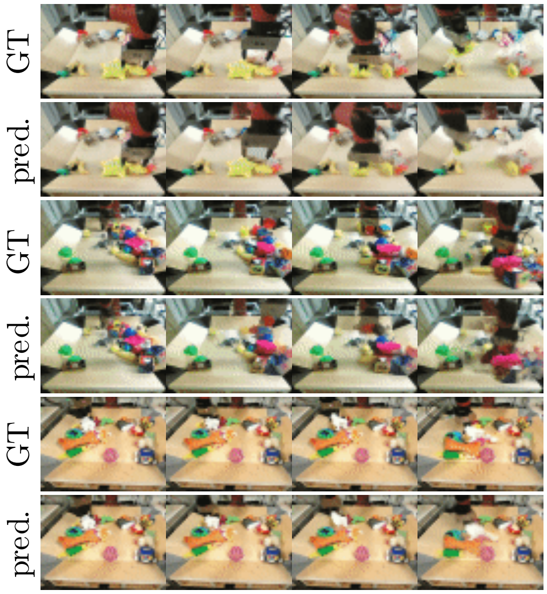


Fig. 1: TO-DO: Overview of visual MPC concept

goal image registration, and image classifiers. Finally, our evaluation demonstrates that these components can be combined to enable a real robot to perform a range of real-world object manipulation tasks from raw pixel observations. Our experiments include manipulation of previously unseen objects, handling multiple objects, pushing objects around obstructions, handling clutter, recovering from large perturbations, and grasping and maneuvering objects to user-specified locations in 3D-space. These results represent a significant advance in the *generality* of skills that can be acquired by a real robot operating on raw pixel values.

This paper builds upon prior conference papers [**?**]; our new contributions include an empirical evaluation of cloth manipulation and/or stochastic models, analysis of each method involving XX, as well as a comprehensive, open-sourced simulation environment to facilitate future research and better reproducibility.

TO-DO: make big VMPC diagramm!!

2 Related Work

2.1 Large-Scale, Self-Supervised Robotic Learning

In a number of recent works large-scale robotic data collection (often using multiple robots) has been the key factor enabling the automatic acquisition of generalizable features and skills. Several prior works have focused on autonomous data collection for individual skills, such as grasping [3], [4] or obstacle avoidance [5], [6]. In contrast to these methods, our approach learns predictive models that can be used to perform a variety of manipulations, and does not require a success measure or reward function during data collection. The prior method [7] sought to learn an inverse model from raw sensory data without any supervision. While these methods demonstrated effective generalization to new objects, they were limited in the complexity of tasks and time-scale at which these tasks could be performed. The method proposed by [7] was able to plan single pokes, and then greedily execute multiple pokes in sequence. We observe a substantial improvement in the length and complexity of manipulations that can be performed with our models.

Kurutach et al. use a InfoGAN model [8] to learn a latent-space describing the environment states in which planning can be performed. The model can be used to obtain a sequence of waypoints in the latent space between the current state and goal-state. However work still remains to be done to achieve a policy that can transition the from one waypoint to the other.

2.2 Sensory Prediction Models

We propose to leverage sensory prediction models, in particular video-prediction, to achieve large-scale self-supervised learning. Relevant related work on video-prediction has been carried out in the context of synthetic video game images [9], [10] and robotic manipulation [11], [12], [13]. Video prediction without actions has been studied for unstructured videos [14], [15], [16] and driving [17], [18].

Several works have sought to use more complex distributions for future images, for example by using autoregressive models [13], [19]. While this often produces sharp predictions, the resulting models are extremely demanding computationally, and have not been applied to real-world robotic control. In this work, we extend video prediction methods that are based on predicting a transformation from the previous image [12], [18]. Prior work has also sought to predict motion directly in 3D, using 3D point clouds obtained from a depth camera [20], requiring point-to-point correspondences over time, which makes it hard to apply to previously unseen objects. Our predictive model is effective for a wide range of real-world object manipulations and does not require 3D depth sensing or point-to-point correspondences between frames. Prior work has also proposed to plan through learned models via differentiation, though not with visual inputs [21]. We instead use a stochastic, sampling-based planning method [22], [23], which we extend to handle a mixture of continuous and discrete actions

3 Visual Model Predictive Control

In this section we define the visual model-predictive control problem formulation. We assume that the user defines a goal for the robot in terms of pixel motion by clicking on the object and a corresponding target location, or by providing one or several demonstrations. Designated pixel positions and demonstrations can also be combined. Based on either of these task definitions we define per-time step cost functions c_t which are computed based on the results of the *action-conditioned* video-prediction model. To find an action sequence $a_{t_0:T}$ for which $c = \sum_t c_t$ over the time steps is minimal we use sampling based planning: A large number of candidate action sequences is sampled and the model's predictions are evaluated using c. The first action of the sequence which achieved lowest cost is applied to the robot.

To render the planning process more efficient we use the cross-entropy method (CEM), a gradient-free optimization procedure that consists of iteratively resampling action sequences and refitting Gaussian distributions to the actions with the best predicted cost. Further details can be found in section 6.

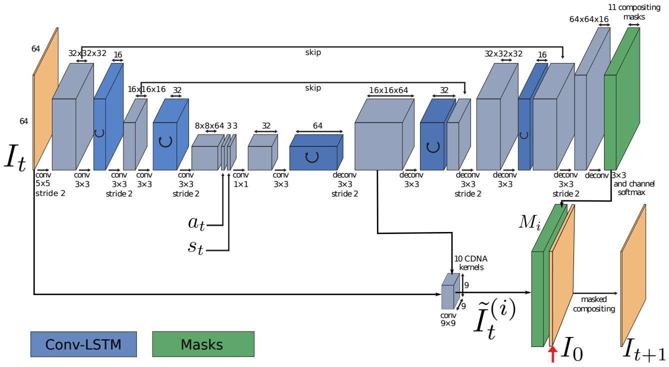


Fig. 2: Simplified SNA model based on Equation (2). The red arrow indicates where the image from the first time step I_0 is concatenated with the transformed images $\tilde{I}_t^{(i)}$ multiplying each channel with a separate mask to produce the predicted frame for step $t+1$. TO-DO: add appflow

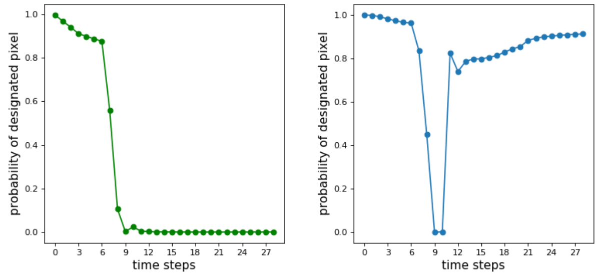


Fig. 3: Predicted probability $P_{d^{(0)}}(t)$ of the designated pixel being at the location of the blue dot indicated in Figure 4 for the DNA model (left) and the SNA model (right).

To improve robustness against imperfect models, the actions are iteratively replanned at each real-world time step $\tau \in \{0, ..., \tau_{max}\}$ following the framework of model-predictive control (MPC): at each real-world step τ, the model is used to plan T steps into the future, and the first action of the plan is executed.

4 Video Prediction for Control

When using a classifier based cost function as detailed in section 5.3 virtually any sensory prediction model can be used. However for the cost function based on user-defined start- and goal pixel positions it is necessary that the model is capable of predicting *where* certain points in the image are moving. The strengths and weaknesses of different costs functions and trade-offs between them are discussed in section **??**.

4.1 Video Prediction via Pixel Transformations

When using visual-MPC with a cost-function based on start- and goal pixel positions, a model is required that can effectively predict the 2-D motion of the selected start pixels $d_0^{(1)}, \ldots, d_0^{(P)}$ up to T steps into the future. To predict the trajectory of these pixels we leverage the model proposed in [12], where this flow prediction capability emerges implicitly, and therefore no external pixel motion supervision is required. Future images are generated by transforming past observations. The model uses stacked convolutional LSTMs that predict a collection of pixel transformations at each time step. In this model, the previous image I_t is transformed using a 2 dimensional flow-field, similar to [24], resampling each pixel at a location specified by the flow field.

Transformed images and the previous *real* image are composited together according to weights obtained from channel-wise selection masks.

The warping transformations can be interpreted as a stochastic transition operator allowing us to make probabilistic predictions about future locations of individual pixels. To predict the future positions of the designated pixels $d^{(i)}$, the same transformations which are used for the images are applied to $P_{t,d^{(i)}}$ such that $P_{t+1,d^{(i)}} = \frac{1}{P_s} \sum_{i=1}^{N} \tilde{P}_{t,d^{(i)}}^{(i)} \mathbf{M}_i$, where $\frac{1}{P_s}$ is a normalizing constant to ensure that $P_{t+1,d^{(i)}}$ adds up to 1 over the spatial dimension of the image. Since this prior model predicts images only based the previous image, it is unable to recover shapes (e.g. objects) after they have been occluded, for example by the robot arm. Therefore, this model is only suitable for planning motions where the user-selected pixels are not occluded during the manipulation, which restricts its use in cluttered environments or with multiple selected pixels. In the next section, we discuss our proposed occlusion-aware model, which lifts this limitation by employing temporal skip connections.

4.2 Skip Connection Neural Advection Model

To enable effective tracking of objects through occlusions, we propose an extension to the video-prediction model proposed in [12] that incorporates temporal skip connections. This model uses a similar multilayer convolutional LSTM structure; with the crucial difference that the transformations now allow transforming pixels not only from the previous but from *all* previous images $I_0, ...I_t$. Transforming from all previous images comes with increased computational cost. However we found that in practice a greatly simplified version of this model, where transformations are applied only to the previous image and the *first image* of the sequence. We will therefore first present the generic model, and then describe the practical simplifications. We refer to this model as the skip connection neural advection model (SNA), since it handles occlusions by copying pixels from prior images in the history such that when a pixel is occluded (e.g., by the robot arm or by another object) it can still reappear later in the sequence. When predicting the next image $\hat{I}_{t+1}$, the generic SNA model transforms each image in the history according to a different transformation and with different masks to produce $\hat{I}_{t+1}$ (see **??** in the appendix):

$$\hat{I}_{t+1} = \sum_{j=t-T}^{t} \sum_{i=1}^{N} \mathbf{M}_{i,j} \tilde{I}_j^{(i)} \tag{1}$$

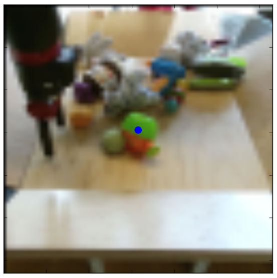


Fig. 4: The blue dot indicates the designated pixel

In the case where $t < T$, negative values of j simply reuse the first image in the sequence. This generic formulation can be computationally expensive, since the number of masks and transformations scales with $T \times N$. A more tractable model, which we found works comparably well in practice in our robotic manipulation setting, assumes that occluded objects are static throughout the prediction horizon. This assumption allows us to dispense the intermediate transformations and only provide a skip connection from the very first image in the sequence, which is also the only real image, since all of the subsequent images are predicted by the model:

$$\hat{I}_{t+1} = I_0 \mathbf{M}_{N+1} + \sum_{i=1}^{N} \tilde{I}_t^{(i)} \mathbf{M}_i \tag{2}$$

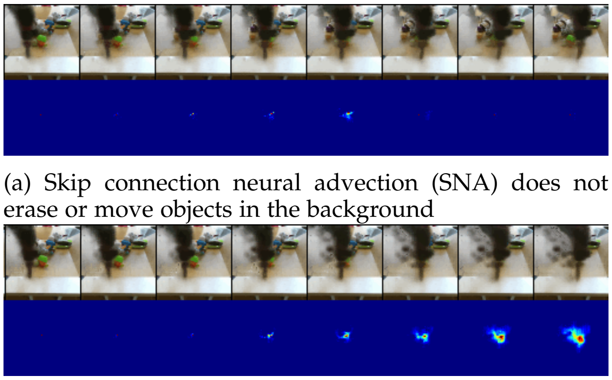
This model only needs to output $N+1$ masks. We observed similar prediction performance when using the original image I_0 instead of the transformed image $\tilde{I}_0$ and therefore use the simplified model in Equation (2) in all of our experiments. We provide an example of the model recovering from occlusion in Figure 5. In the figure, the arm moves in front of the designated pixel, marked in blue in Figure 4. The graphs in Figure 3 show the predicted probability of the designated pixel, which is stationary during the entire motion, being at its original position at each step. Precisely when the arm occludes the designated pixel, the pixel's probability of being at this point decreases. This indicates that the model is 'unsure' where this pixel is. When the arm unoccludes the designated pixel, it should become visible again, and the probability of the designated pixel being at its original position should go up. In the case of the DNA model and its variants [12], the probability mass does not increase after the object reappears. This is because the DNA model cannot recover information about object that it has 'overwritten' during its predictions.

Consequently these models give wrong predictions when the designated pixel becomes occluded often causing the model to predict that the pixel *moves with the arm*. We identified this as one of the major causes of planning failure when using these prior models. By contrast our SNA model predicts that the occluded object will not move, which is indicated by a sharp increase in the probability value evaluated at the original position of the pixel after becoming unoccluded (see Figure 3).

5 Planning Cost Functions

5.1 Pixel-Distance based Cost

A convenient way to define a robot task is by choosing one or more pixels in the robot's camera view and choosing a destination where each pixel should be moved. For example, the user might select a pixel on an object and ask the robot to move it 10 cm to the left. Formally, the user specifies P source pixel locations $d_0^{(1)}, \dots, d_0^{(P)}$ in the initial image I_0, and P goal locations $g^{(1)}, \dots, g^{(P)}$. The source and goal pixel locations are denoted by the coordinates $(x_d^{(i)}, y_d^{(i)})$ and $(x_g^{(i)}, y_g^{(i)})$. Given a goal, the robot plans for a sequence of actions $\mathbf{a}_{1:T}$ over T time steps, where T is the planning horizon. For this type of task definition the problem is formulated as the minimization of a cost function c which depends on the predicted pixel positions $d_t^{(j)}$. The planner makes use of a learned model that predicts a distribution over the pixel position by internally predicting pixel transformations. Given a distribution over pixel positions $P_{t_0,d^{(i)}} \in \mathbb{R}^{H \times W}$ at time $t = 0$, the model predicts distributions over its positions $P_{t,d^{(i)}}$ at time $t \in \{1, \dots, T\}$. The optimizer then finds the action sequence $a_{t_0:T}$ for which the sum of the costs c_t over the time steps is minimal. c_t is defined as the expected euclidean distance between to the goal point $g^{(i)}$ (which is straight-forward to calculate from $P_{t,d^{(i)}}$ and $g^{(i)}$).



(a) Skip connection neural advection (SNA) does not erase or move objects in the background

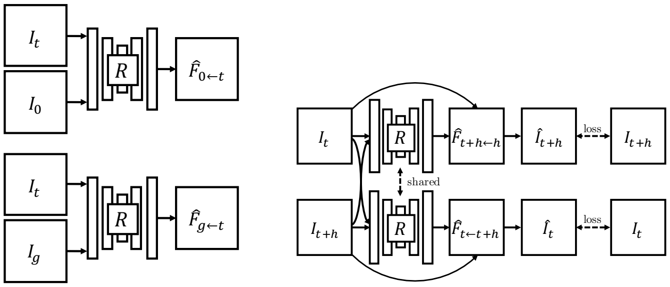
(b) Standard DNA [23] exhibits undesirable movement of the distribution $P_d(t)$ and erases the background

Fig. 5: Top rows: Predicted images of arm moving *in front of* green object with designated pixel (as indicated in Figure 4). Bottom rows: Predicted probability distributions $P_d(t)$ of designated pixel obtained by repeatedly applying transformations.

The expected distance to the goal provides smoother planning objective that makes used of the uncertainty estimates of the predictor enabling complex, longer-horizon tasks. This cost function encourages the movement of the designated objects in the right direction for each step of the execution, regardless of whether the g position can be reached within T time steps. For multi-objective tasks with multiple designated pixels $d^{(i)}$ the costs are summed to together (and optionally weighting them according to a scheme discussed in Section 5.2).

5.2 Registration-based Cost

For the type of cost function introduced in the previous section it necessary to know the initial locations $d_0^{(1)}, \dots, d_0^{(P)}$ at each timesteps. However when the tobot interacts with an object, the position of that object may change in unexpected ways, and therefore it is crucial that the system can update its belief of where the target object is. Prior work on visual MPC lacked this capability. To enable the agent to "keep retrying" indefinitely until the task is accomplished we propose to use an image-to-image registration approach to find the target object in the current frame. We show that this tracking capability can be learned completely unsupervised from the data collected by the robot autonomously. As one might expect the self-supervised tracking approach works better for images that are closer to each other and sometimes fails when the entities in the image are too far apart. To


I_t
I_0
R
$\hat{F}_{0\leftarrow t}$
I_t
I_g
R
$\hat{F}_{g\leftarrow t}$
I_t
R
$\hat{F}_{t+h\leftarrow h}$
$\hat{I}_{t+h}$
loss
I_{t+h}
shared
I_{t+h}
R
$\hat{F}_{t\leftarrow t+h}$
$\hat{I}_t$
loss
I_t

(a) Testing usage. (b) Training usage.

Fig. 6: (a) At test time the registration network registers the current image I_t to the start image I_0 (top) and goal image I_g (bottom), inferring the flow-fields $\hat{F}_{0\leftarrow t}$ and $\hat{F}_{g\leftarrow t}$. (b) The registration network is trained by warping images from randomly selected timesteps along a trajectory to each other.

increase accuracy and robustness in addition to the starting image our method can register to a goal-image. The goal image is optional and can be provided by a human or through demonstration.

5.2.1 Test time procedure

We will first describe the architecture at test time (see Figure 6). The start and goal images I_0 and I_g are passed into the registration network R which returns flow maps:

$$\hat{F}_{0\leftarrow t} = R(I_t, I_0) \qquad \hat{F}_{g\leftarrow t} = R(I_t, I_g) \tag{3}$$

The flow maps warp the image of the current time step to the start image I_0, and $\hat{F}_{g\leftarrow t}$ warps from I_t to I_g (see Figure **??** for an illustration):

$$\hat{I}_0 = \hat{F}_{0\leftarrow t} \diamond I_t \qquad \hat{I}_g = \hat{F}_{g\leftarrow t} \diamond I_t \tag{4}$$

Here, $\diamond$ denote a bilinear sampling operator which interpolates a position in an image bilinearly with respect to a location (x, y) in the image. In addition to start and goal images, the user needs to specify one or several designated pixel positions d_0 in the start image and the corresponding pixels locations d_g in the goal image (the goal image is optional) to define the task. While the registration network is trained to perform a global registration between the images, we only evaluate it at the points d_0 and d_g chosen by the user. This results in a cost function that ignores distractors. R takes in a pair of images and outputs a flow-map warping points in one to the other. Thus, for a current image I_t, $\hat{F}_{0\leftarrow t}$ puts it in correspondence with I_0, and $\hat{F}_{g\leftarrow t}$ puts it in correspondence with I_g. The registration network is then used to find the pixel locations corresponding to d_0 and d_g in the current frame:

$$\hat{d}_{0,t} = d_0 + \hat{F}_{0\leftarrow t}(d_0) \qquad \hat{d}_{g,t} = d_g + \hat{F}_{g\leftarrow t}(d_g) \tag{5}$$

Figure 7 visualizes the tracking results during a pushing task. On the left we show the start image and goal image provided by the user at the beginning of the trajectory. The first row shows the video, with the estimated location of d_0 marked in red and the estimated location of d_g marked in blue. In this example, registration succeeds with respect to the first image succeeds for the first 7 steps but fails after that. However registration with respect to the goal image succeeds for the last 3 steps. Thus, there is enough information at each time step to determine the cost, which we discuss in detail in the next section.

5.2.2 Train time procedure

The registration network R is trained to find correspondences between two images by warping one image to the other. The network is trained on the same data as the video-prediction model, but it does not share parameters with it.[1] This approach is similar to the optic flow method proposed by [25]. However, unlike this prior work, our method computes registrations for frames that might be many time steps apart, and the goal is not to extract optic flow, but rather to determine correspondences between potential highly distinct images. For training, two images are sampled at random times steps t and $t + h$ along the trajectory and the images are warped to each other in both directions.

$$\hat{I}_t = \hat{F}_{t\leftarrow t+h} \diamond I_{t+h} \qquad \hat{I}_{t+h} = \hat{F}_{t+h\leftarrow t} \diamond I_t \tag{6}$$

The network, which outputs $\hat{F}_{t\leftarrow t+h}$ and $\hat{F}_{t+h\leftarrow t}$ (see Figure 6), is trained to minimize the photometric distance between $\hat{I}_t$ and I_t and $\hat{I}_{t+h}$ and I_{t+h}, in addition to a smoothness regularizer that penalizes abrupt changes in the outputted flow-field. The details of this loss function follow prior work [25]. We found that gradually increasing the temporal distance h between the images during training yielded better final accuracy, as it created a curriculum for the model. The network R is implemented as a fully convolutional network which takes in as input the two images stacked together along the channel dimension. We use three convolutional layers each followed by a bilinear downsampling operation followed by three layers of convolution each followed by a bilinear upsampling operation (all convolutions use stride 1). In this way we avoid artifacts due to strided convolutions and deconvolutions.

5.3 Classifier-based Cost

An alternative way to define the cost function is with a goal classifier. This type of cost function is particularly well-suited for tasks that can be completed in multiple ways. For example, for a task of rearranging a pair objects into relative positions, i.e. pushing the first object to the left of the second object, the absolute positions of the objects do not matter. A classifier-based cost function allows the planner to discover any of the possible goal states.

Formally, we consider a goal classifier $\hat{y} = f(\mathbf{o})$, where $\mathbf{o}$ denotes the image observation, and $\hat{y} \in [0, 1]$ indicates the predicted probability of the observation being of a successful outcome of the task. Our objective is to infer a classifier for a new task $\mathcal{T}_j$ from a few positive examples of success, which are easy for a user to provide and encode the minimal information needed to convey a task. In other words, given a dataset $\mathcal{D}_j^+$ of K examples of successful end states for a new task $\mathcal{T}_j$: $\mathcal{D}_j := \{(\mathbf{o}_k, 1) | k = 1...K\}_j$, our goal is to infer a classifier for task $\mathcal{T}_j$.

5.4 Which cost function is best?

adsf

[1] in principle sharing parameters with the video-prediction model might be beneficial, however this is left for future work

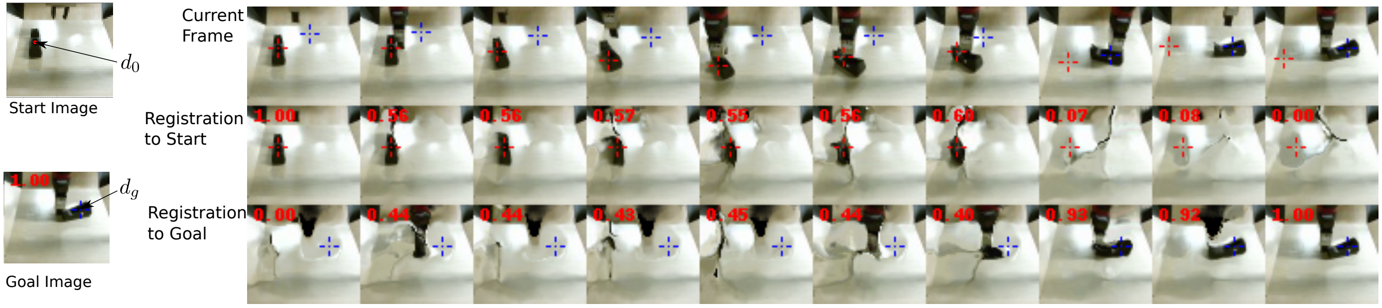


Fig. 7: Outputs of registration network. The first row shows images from a trajectory executed by the robot, the second shows each image warped to the initial image via registration, and the third shows the same for the goal image. A successful registration in this visualization would result in images that closely resemble the start (or goal). In these images, the locations where the designated pixel of the start image d_0 and the goal image d_g are found is marked with red and blue crosses, respectively. It can be seen that, for the registration to the start image (red cross) the object is tracked for the first 7 frames, while the registration to the goal image (blue cross) succeeds for the last 3 time steps. The numbers in red indicate the trade off factors λ between the views and are used as weighting factors for the planning cost.

5.4.1 Meta-Learning for Few-Shot Goal Inference

To solve the above problem, we propose learning a few-shot classifier that can infer the goal of a new task from a small set of goal examples, allowing the user to define a task from goal images alone.

To train the few-shot classifier, we build upon model-agnostic meta-learning (MAML) [26], which learns initial parameters θ for model f that can efficiently adapt to a new task with one or a few steps of gradient descent. [27] proposed an extension of MAML, referred to as concept acquisition through meta-learning (CAML), for learning new concepts from positive examples alone. We apply CAML to the setting of acquiring goal classifiers from positive examples.

5.4.2 Test time procedure

At test time, the user provides a dataset $\mathcal{D}_j^+$ of K examples of successful end states for a new task $\mathcal{T}_j$: $\mathcal{D}_j := \{(\mathbf{o}_k, 1)|k = 1...K\}_j$, which are then used to infer a task-specific goal classifier C_j. In particular, the meta-learned parameters θ are updated through gradient descent to adapt to task $\mathcal{T}_j$:

$$C_j(\mathbf{o}) = f(\mathbf{o}; \theta_j') = f\big(\mathbf{o}; \theta - \alpha \nabla_\theta \sum_{(\mathbf{o}_n, y_n) \in \mathcal{D}_j^+} \mathcal{L}(y_n, f(\mathbf{o}_n; \theta)\big)$$

where $\mathcal{L}$ is the cross-entropy loss function, α is the step size, and θ' denotes the parameters updated through gradient descent on task $\mathcal{T}_j$.

During planning, the learned classifier C_j takes as input an image generated by the video-prediction model and outputs the predicted probability of the goal being achieved for the task. Concretely, the cost is defined as the negative of the classifier prediction.

5.4.3 Train time procedure

During meta-training, we explicitly train for the ability to infer goal classifiers for the set of training tasks, $\{\mathcal{T}_i\}$. We assume a small dataset $\mathcal{D}_i$ for each task $\mathcal{T}_i$, consisting of both positive and negative examples: $\mathcal{D}_i := \{(\mathbf{o}_n, y_n)|n = 1...N\}_i$. To learn the initial parameters θ, we optimize the following objective:

$$\min_\theta \sum_i \sum_{(\mathbf{o}_n, y_n) \in \mathcal{D}_i^{\text{test}}} \mathcal{L}(y_n, f(\mathbf{o}_n; \theta_i'))$$

We optimize this objective using Adam [28] on the initial parameters θ. In our experiments, our classifier is represented by a convolutional neural network, consisting of three convolutional layers, each followed by layer normalization and a ReLU non-linearity. After the final convolutional layer, a spatial soft-argmax operation extracts spatial feature points, which are then passed through fully-connected layers.

6 Trajectory Optimizer

The role of the optimizer is to find actions sequences $a_{t:t+T}$ which minimize the sum of the costs $c_{t:t+T}$.

In the visual MPC setting the action sequences found by the optimizer can be very different between execution times steps τ (not to be confused with prediction time steps t). For example at one time step τ the optimizer might find a pushing action leading towards the goal and in the next time step $\tau + 1$ it determines a grasping action to be optimal to reach the goal. Naive replanning at every time step can then result in alternating between a pushing and a grasping attempt indefinitely causing the agent to get stuck and not making any progress towards to goal.

We show that we can resolve this problem by modifying the sampling distribution of the first iteration of CEM so that the optimizer commits to the plan found in the previous time step. In prior work [29] the sampling distribution at first iteration of CEM is chosen to be a Gaussian with diagonal covariance matrix and zero mean. We instead use the best action sequence found in the optimization of the previous time step as the mean. Since this action sequence is optimized for the previous time step we only use the values $a_{1:T}$ and omit the first action, where T is the prediction horizon. To sample actions close to the action sequence from the previous time step we reduce the entries of the diagonal covariance matrix for the first $T-1$ time steps. It is crucial that the last entry of the covariance matrix at the end of the horizon is not reduced otherwise no exploration could happen for the last time step causing poor performance at later time steps.

7 Experimental evaluation

7.1 Evaluating Skip-connection Neural Advection

Our experimental evaluation compares the proposed occlusion-aware SNA video prediction model, as well as the improved cost function for planning, with a previously proposed model based on dynamic neural advection (DNA) [23]. We use a Sawyer robot, shown in **??**, to push a variety of objects in a tabletop setting. In the appendix, we include a details on hyperparameters, analysis of sample complexity, and a discussion of robustness and limitations. We evaluate long pushes and multi-objective tasks where one object must be pushed without disturbing another. The supplementary video and links to the code and data are available at https://sites.google.com/view/sna-visual-mpc

Figure 8 shows an example task for the pushing benchmark. We collected 20 trajectories with 3 novel objects and 1 training object. Table 3 shows the results for the pushing benchmark. The column *distance* refers to the mean distance between the goal pixel and the designated pixel at the final time-step. The column *improvement* indicates how much the designated pixel of the objects could be moved closer to their goal (or further away for negative values) compared to the starting location. The true locations of the designated pixels after pushing were annotated by a human.

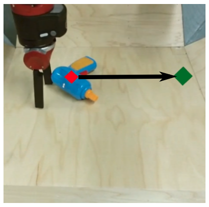


Fig. 8: Pushing task. The designated pixel (red diamond) needs to be pushed to the green circle.

The results in Table 3 show that our proposed planning cost in **??** substantially outperforms the planning cost used in prior work [23]. The performance of the SNA model in these experiments is comparable to the prior DNA model [23] when both use the new planning cost, since this task does not involve any occlusions. Although the DNA model has a slightly better mean distance compared to SNA, it is well within the standard deviation, suggesting that the difference is not significant.

To examine how well each approach can handle occlusions, we devised a second task that requires the robot to push one object, while keeping another object stationary. When the stationary object is in the way, the robot must move the goal object around it, as shown in Figure 9 on the left. While doing this, the gripper may occlude the stationary object, and the task can only be performed successfully if the model can make accurate predictions through this occlusion. These tasks are specified by picking one pixel on the target object, and one on the obstacle. The obstacle is commanded to remain stationary, while the target object destination location is chosen on the other side of the obstacle.

We used four different object arrangements, with two training objects and two objects that were unseen during training. We found that, in most of the cases, the SNA model was able to find a valid trajectory, while the prior DNA model was mostly unable to find a solution. Figure 9 shows an example of the SNA model successfully predicting the position of the obstacle through an occlusion and finding a trajectory that avoids the obstacle. These findings are reflected by our quantitative results shown in Table 2, indicating the importance of temporal skip connections.

7.2 Evaluating Closed loop Visual MPC

Our experimental evaluation, conducted using two Rethink Sawyer robotic manipulators, evaluate the ability of our method to learn both prehensile and non-prehensile object relocation tasks entirely through autonomously collected data and self-supervision. In particular, we aim to answer the following questions: (1) How does our MPC approach with self-supervised goal image registration compare to alternative cost functions, such as off-the-shelf tracking, forward prediction via flow-based models, and pixel difference costs? (2) When the robot can continuously retry a task with goal image registration, how much is the success rate for object relocation tasks improved? (3) Can we learn predictive models that enable both non-prehensile and prehensile object manipulation? We also present additional experimental comparisons in a simulated environment. Videos and visualizations can be found on this webpage: https://sites.google.com/view/robustness-via-retrying.

7.2.1 Real-World Experiments

To train both our prediction and registration models, we collected 20,000 trajectories of pushing motions and 15,000 trajectories with gripper control, where the robot was allowed to randomly move and pick up objects. The data collection process is fully autonomous, requiring human intervention only to replace and change out the objects in front of the robot. The action space consisted of Cartesian movements along the x, y, and z axes, and changes in azimuthal orientation of the gripper. For evaluation, we selected novel objects that were never seen during training. The evaluation tasks required the robot to move objects in its environment from a starting state to a goal configuration, and performance was evaluated by measuring the distance between the final object position and goal position. In all experiments, the maximum episode length was 50 time steps.

7.2.2 Pushing with retrying

In the first experiment, we aim to evaluate the performance of different visual MPC cost functions, including our proposed self-supervised registration cost. For this experiment, we disable the gripper control, which requires the robot to push objects to the target. We evaluate 20 different pushing tasks that require pushing an object to a target position. For comparisons, we include a baseline where the object is tracked using the "multiple instance learning tracker" MIL [30] in OpenCV. Note that our method does not have any prior knowledge of objects – it is only provided with the position of one designed pixel in the initial and goal images, and must use the learned model to infer that this pixel belongs to an object that can be moved by the robot. We also compare to the visual MPC method proposed by [29], which does not track the object explicitly, but relies on the flow-based video prediction model to keep track of the designated pixel, which we call "predictor propagation." We also tested a method where the visual MPC cost is calculated

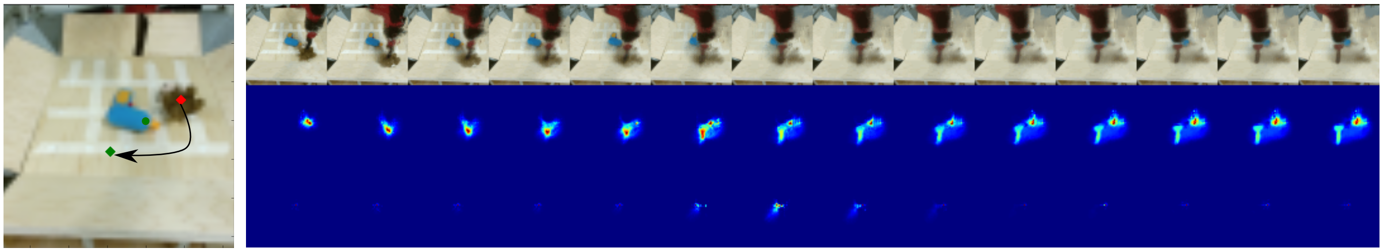


Fig. 9: Left: Task setup with green dot marking the obstacle. Right, first row: the predicted frames generated by SNA. Second row: the probability distribution of the designated pixel on the *moving* object (brown stuffed animal). Note that part of the distribution shifts down and left, which is the indicated goal. Third row: the probability distribution of the designated pixel on the obstacle-object (blue power drill). Although the distribution increases in entropy during the occlusion (in the middle), it then recovers and remains on its original position.

	seen		unseen	
	distance mean, std.	improvement mean, std.	dist. mean, std.	improvement mean, std.
random actions	29.4 ± 3.2	-0.85 ± 2.3	N/A	N/A
DNA with distance metric as in [23]	25.9 ± 12.2	5.2± 13.7	24.6±10.6	2.1±12.4
DNA with our exp. dist. metric eqn. **??**	**9.2 ±7.4**	**15. ± 11.5**	**17.5 ± 10.2**	**8.3 ± 11.8**
SNA exp. dist. metric eqn. **??** (ours)	13.0 ± 5.6	12.4 ± 8.8	18.18 ± 9.5	7.7 ± 10.5

TABLE 1: Results of the pushing benchmark on 20 different object/goal configurations. Units are pixels in the 64x64 images.

	seen		unseen	
	moved imp. mean, std.	stationary imp. mean, std.	moved imp. mean, std.	stationary imp. mean, std.
Dynamic Neural Advection, DNA [23]	3.5 ±4.94	-1.4 ± 0.39	0.83 ±1.13	-1.1 ± 0.9
Skip Con. Neural Advection, SNA (ours)	**8.16 ± 2.9**	**-0.9 ±0.7**	**10.6 ± 3.7**	**-1.5 ± 0.9**

TABLE 2: Results for multi-objective pushing on 8 object/goal configurations with 2 seen and 2 novel objects. Values indicate improvement in distance from starting position, higher is better. Units are pixels in the 64x64 images.

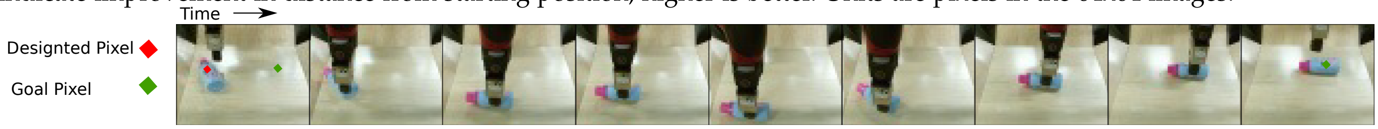


Fig. 10: Applying our method to a pushing task. In the first 3 time instants the object behaves unexpectedly, moving down. The tracking then allows the robot to retry, allowing it to eventually bring the object to the goal.

	Short	Long
Visual MPC + predictor propagation	83%	20%
Visual MPC + OpenCV tracking	83%	45%
Visual MPC + registration network (Ours)	83%	**66%**

TABLE 3: Success rate for long-distance pushing benchmark with 20 different object/goal configurations and short-distance benchmark with 15 object/goal configurations. Success is defined as bringing the object closer than 15 pixels to the goal, where the complete image has size 48x64.

by directly evaluating the error between the predicted image and goal image, but we found that this approach was unable to make meaningful progress in moving the object to the target. Instead it resulted in a policy that always moves the arm to the position indicted in the target image, as the arm accounts for the majority of the movable pixel mass in the image. This can be interpreted as undesirable local minimum which is not present in tracking based cost we propose.

?? illustrates that our approach not only outperforms prior work [29], but also outperforms the hand-designed object tracker [30]. This is due to the fact that using our learned registration the agent is more frequently able to successfully recovering from situations where the object behaved differently than predicted the model predicted (see Figure 10).

7.2.3 Combined prehensile and non-prehensile manipulation.

In the setting where the gripper is enabled it is part of the task to decide whether to solve a task by prehensile or non-prehensile manipulation. Similarly to the pushing setting we perform a benchmark where we define a set of 20 object relocation tasks and measure the final distance between the object and the target at the end of the episode. Interestingly we observe that in the majority of the cases the agent decides to grasp the object, as can be seen in the supplementary video. **??** shows that the performance of our method is comparable with the performance of OpenCV tracking.

7.3 Evaluating Classifier-based cost function

We study a visual object arrangement task, where different goals correspond to different relative arrangements of a

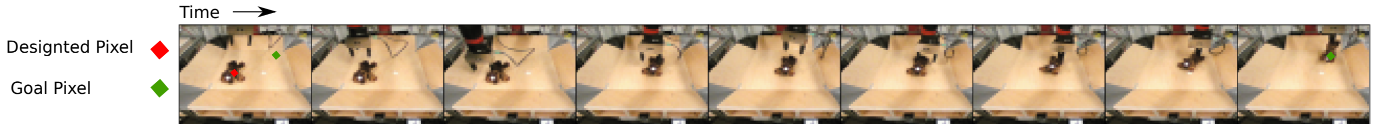


Fig. 11: Retrying behaviour of our method combining prehensile and non-prehensile manipulation. In the first 4 time instants shown the agent pushes the object. It then loses the object, and decides to grasp it pulling it all the way to the goal. Retrying is enabled by applying the learned registration to both camera views (here we only show the front view).

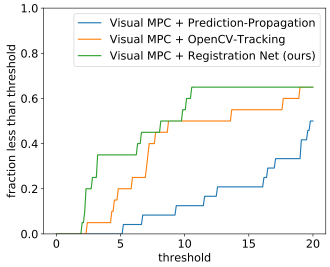


Fig. 12: Results for long pushing tasks with 20 objects not seen during training, showing fraction of runs where final distance is lower than threshold. Our method shows a clear gains over OpenCV tracking and predictor propagation.

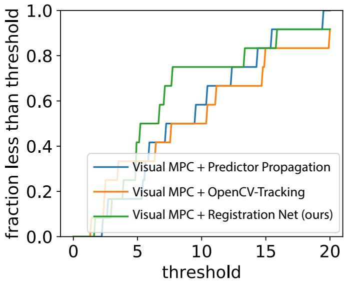


Fig. 13: Results for short pushing tasks. Fraction of runs where final distance is lower than threshold.

pair of objects. We evaluate our learned classifier on the predictions made by the video prediction model and derive the cost used for planning from the predicted probability of success.

7.3.1 Real-World Experiments

To collect data for meta-training the classifier, we randomly select a pair of objects from our set of training objects, and position them into many different relative positions, recording the image for each configuration. One task corresponds to a particular relative positioning of two objects, e.g. the first object to the left of the second, and we construct positive and negative examples for this task by labeling the aforementioned images. We randomly position the arm in each image, as it is not a determiner of task success. A good objective should ignore the position of the arm. We also include randomly-positioned distractor objects in about a third of the collected images.

We evaluate all approaches in three different experimental settings. In the first setting, the goal is to arrange two objects into a specified relative arrangement. The second setting is the same, but with distractor objects present. In the final, most challenging setting, the goal is to achieve two tasks in sequence. We provide positive examples for both tasks, infer the classifier for both task, perform MPC for the first task until completion, followed by MPC for the second task. To evaluate the ability to generalize to new goals and settings, we use novel, held-out objects for all of the task and distractor objects in our evaluation.

We qualitatively visualize the evaluation in Figure **??**. On the left, we show a subset of the five images provided to illustrate the task(s), and on the left, we show the motions performed by the robot. We see that the robot is able to execute motions which lead to a correct relative positioning of the objects. We quantitatively evaluate each method across 20 tasks, including 10 unique object pairs. The results, shown in Figure **??**, indicate that prior methods for learning distance metrics struggle to infer the goal of the task, while our approach leads to substantially more successful behavior on average.

8 Discussion

9 Conclusion

The conclusion goes here.

Appendix A
Simulated Experiments

A.0.1 Simulated Experiments

In order to provide a more controlled comparison, we also set up a realistic simulation environment using MuJoCo [31], which includes a robotic manipulator controlled via Cartesian position control, similar to our real world setup, pushing randomly-generated L-shaped objects with random colors (see details in supplementary materials). We trained the same video prediction model in this environment, and set up 50 evaluation tasks where blocks must be pushed to target locations with maximum episode lengths of 120 steps. We compare our proposed registration-based method, "predictor propagation," and ground-truth registration obtained from the simulator, which provides an oracle upper bound on registration performance. **??** shows the results of this simulated evaluation, where the x-axis shows different distance thresholds, and the y-axis shows the fraction of evaluation scenarios where each method pushed the object within that threshold. We can see that, for thresholds around 0.1, our method drastically outperforms predictor propagation (i.e., prior work [29]), and has a relatively modest gap in performance against ground-truth tracking. This indicates that our registration method is highly effective in guiding visual MPC, despite being entirely self-supervised.

Acknowledgments

The authors would like to thank...

References

[1] S. Levine, C. Finn, T. Darrell, and P. Abbeel, "End-to-end learning of deep visuomotor policies," *Journal of Machine Learning Research (JMLR)*, 2016.

[2] C. Finn, X. Y. Tan, Y. Duan, T. Darrell, S. Levine, and P. Abbeel, "Deep spatial autoencoders for visuomotor learning," in *Robotics and Automation (ICRA), 2016 IEEE International Conference on*. IEEE, 2016.

[3] L. Pinto and A. Gupta, "Supersizing self-supervision: Learning to grasp from 50k tries and 700 robot hours," in *International Conference on Robotics and Automation (ICRA)*, 2016.

[4] S. Levine, P. Pastor, A. Krizhevsky, J. Ibarz, and D. Quillen, "Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection," *International Journal of Robotics Research (IJRR)*, 2016.

[5] G. Kahn, A. Villaflor, V. Pong, P. Abbeel, and S. Levine, "Uncertainty-aware reinforcement learning for collision avoidance," *arXiv preprint arXiv:1702.01182*, 2017.

[6] D. Gandhi, L. Pinto, and A. Gupta, "Learning to fly by crashing," *arXiv preprint arXiv:1704.05588*, 2017.

[7] P. Agrawal, A. V. Nair, P. Abbeel, J. Malik, and S. Levine, "Learning to poke by poking: Experiential learning of intuitive physics," in *Advances in Neural Information Processing Systems*, 2016.

[8] T. Kurutach, A. Tamar, G. Yang, S. Russell, and P. Abbeel, "Learning plannable representations with causal infogan," *arXiv preprint arXiv:1807.09341*, 2018.

[9] J. Oh, X. Guo, H. Lee, R. L. Lewis, and S. Singh, "Action-conditional video prediction using deep networks in atari games," in *Advances in Neural Information Processing Systems*, 2015.

[10] S. Chiappa, S. Racaniere, D. Wierstra, and S. Mohamed, "Recurrent environment simulators," *arXiv preprint arXiv:1704.02254*, 2017.

[11] B. Boots, A. Byravan, and D. Fox, "Learning predictive models of a depth camera & manipulator from raw execution traces," in *International Conference on Robotics and Automation (ICRA)*. IEEE, 2014.

[12] C. Finn, I. Goodfellow, and S. Levine, "Unsupervised learning for physical interaction through video prediction," in *Neural Information Processing Systems (NIPS)*, 2016.

[13] N. Kalchbrenner, A. v. d. Oord, K. Simonyan, I. Danihelka, O. Vinyals, A. Graves, and K. Kavukcuoglu, "Video pixel networks," *arXiv preprint arXiv:1610.00527*, 2016.

[14] M. Mathieu, C. Couprie, and Y. LeCun, "Deep multi-scale video prediction beyond mean square error," *International Conference on Learning Representations (ICLR)*, 2016.

[15] S. Xingjian, Z. Chen, H. Wang, D.-Y. Yeung, W.-K. Wong, and W.-c. Woo, "Convolutional lstm network: A machine learning approach for precipitation nowcasting," in *Advances in neural information processing systems*, 2015.

[16] C. Vondrick, H. Pirsiavash, and A. Torralba, "Generating videos with scene dynamics," in *Advances In Neural Information Processing Systems*, 2016.

[17] W. Lotter, G. Kreiman, and D. Cox, "Deep predictive coding networks for video prediction and unsupervised learning," *International Conference on Learning Representations (ICLR)*, 2017.

[18] B. De Brabandere, X. Jia, T. Tuytelaars, and L. Van Gool, "Dynamic filter networks," in *Neural Information Processing Systems (NIPS)*, 2016.

[19] S. Reed, A. v. d. Oord, N. Kalchbrenner, S. G. Colmenarejo, Z. Wang, D. Belov, and N. de Freitas, "Parallel multiscale autoregressive density estimation," *arXiv preprint arXiv:1703.03664*, 2017.

[20] A. Byravan and D. Fox, "Se3-nets: Learning rigid body motion using deep neural networks," *arXiv preprint arXiv:1606.02378*, 2016.

[21] I. Lenz and A. Saxena, "Deepmpc: Learning deep latent features for model predictive control," in *In RSS*. Citeseer, 2015.

[22] R. Y. Rubinstein and D. P. Kroese, *The cross-entropy method: a unified approach to combinatorial optimization, Monte-Carlo simulation and machine learning*. Springer Science & Business Media, 2013.

[23] C. Finn and S. Levine, "Deep visual foresight for planning robot motion," in *International Conference on Robotics and Automation (ICRA)*, 2017.

[24] T. Zhou, S. Tulsiani, W. Sun, J. Malik, and A. A. Efros, "View synthesis by appearance flow," in *European conference on computer vision*. Springer, 2016, pp. 286–301.

[25] S. Meister, J. Hur, and S. Roth, "Unflow: Unsupervised learning of optical flow with a bidirectional census loss," *arXiv preprint arXiv:1711.07837*, 2017.

[26] C. Finn, P. Abbeel, and S. Levine, "Model-agnostic meta-learning for fast adaptation of deep networks," *International Conference on Machine Learning (ICML)*, 2017.

[27] E. Grant, C. Finn, J. Peterson, J. Abbott, S. Levine, T. Griffiths, and T. Darrell, "Concept acquisition via meta-learning: Few-shot learning from positive examples," in *NIPS Workshop on Cognitively-Informed Artificial Intelligence*, 2017.

[28] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *CoRR*, vol. abs/1412.6980, 2014.

[29] F. Ebert, C. Finn, A. X. Lee, and S. Levine, "Self-supervised visual planning with temporal skip connections," *CoRR*, vol. abs/1710.05268, 2017. [Online]. Available: http://arxiv.org/abs/1710.05268

[30] B. Babenko, M.-H. Yang, and S. Belongie, "Visual tracking with online multiple instance learning," in *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*. IEEE, 2009, pp. 983–990.

[31] E. Todorov, T. Erez, and Y. Tassa, "Mujoco: A physics engine for model-based control," in *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*. IEEE, 2012, pp. 5026–5033.

PLACE
PHOTO
HERE

Michael Shell Biography text here.

John Doe Biography text here.

Jane Doe Biography text here.