Data-Driven Visual Forecasting

Jacob Charles Walker

April 2018
CMU-RI-TR-18-12

The Robotics Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Robotics

Acknowledgements

One of the most crucial factors for success in graduate school are the advisors—they provide the foundation for a student's career. I therefore must first thank my advisors Martial Hebert and Abhinav Gupta for their support and aid through this academic journey. I appreciate their presence, effort, and guidance over the last five years. In addition, I also am grateful for my thesis committee. Thank you, Ruslan Salakhutdinov and David Forsyth, for your time, comments, and feedback on this thesis. Your assistance has been indispensable. Another thanks goes to Bart Nabbe and Tali Dekel for mentoring my summer internships. Your collaborations have broadened my perspective. In addition, I thank Lynnetta Miller, Christine Downey, and Suzanne Muth for their aid.

I also cannot forget the support and camaraderie from the colleagues in my lab, the Robotics Institute, and Carnegie Mellon University in general. Thank you Xinlei Chen, David Fouhey, Abhinav Shrivistava, Ishan Misra, Carl Doersch, Ada Zhang, Kenneth Marino, Aayush Bansal, Achal Dave, Peiyun Wang, Rohit Girdhar, Gunnar Siggurdson, Rick Goldstein, Humphrey Hu, Leo Keselman, Xiaolong Wang, Nadine Chang, Brian Okorn, Nick Rhinehart, Robin Zhou, and Shaurya Shankar. Whether it's critiquing my talks and papers, discussing research ideas, or merely meeting up for drinks at the Yard, your company is valued beyond measure.

Abstract

Understanding the temporal dimension of images is a fundamental part of computer vision. Humans are able to interpret how the entities in an image will change over time. However, it has only been relatively recently that researchers have focused on visual forecasting—getting machines to anticipate events in the visual world before they actually happen. This aspect of vision has many practical implications for tasks ranging from human-computer interaction to anomaly detection. In addition, temporal prediction can serve as a task for representation learning, useful for various other recognition problems.

In this thesis, we focus on visual forecasting that is data-driven, self-supervised, and relies on little to no explicit semantic information. Towards this goal, we explore prediction at different timeframes. We first consider predicting instantaneous pixel motion—optical flow. We apply convolutional neural networks to predict optical flow in static images. We then extend this idea to a longer timeframe, generalizing to pixel trajectory prediction in space-time. We incorporate models such as variational autoencoders to generate future possible motions in the scene. After this, we consider a mid-level element approach to forecasting. By combining a Markovian reasoning framework with an intermediate representation, we are able to forecast events over longer timescales.

This dissertation then builds upon these ideas towards structured representations for visual forecasting. Specifically, we aim to reason about the future of images in a structured state space. Instead of directly predicting events in a low-level feature space such as pixels or motion, we forecast events in a higher level representation that is still visually meaningful. This approach confers a number of advantages. It is not restricted by explicit timescales like motion-based approaches, and, unlike direct pixel-based approaches, predictions are less likely to “fall off” the manifold of the true visual world.

This research was supported by NSF grant IIS-1227495. We also thank Nvidia for donated GPUs.

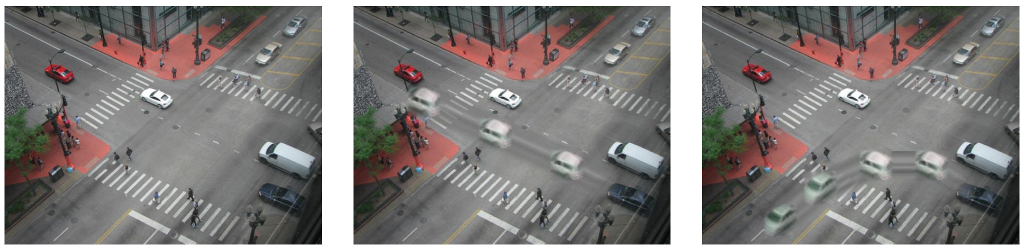
Contents

List of Figures

List of Tables

Chapter 1

Introduction and Summary



(a) Input Image (b) Predicted Event 1 (c) Predicted Event 2

Figure 1.1: Consider this traffic scene. We as humans can often anticipate and visualize multiple possible future outcomes. (a). The blue car moving forward (b) and turning left (c) are two such possibilities. In this thesis, we aim to endow computer vision systems with this capability—specifically with models which learn from unsupervised data.

Consider the image shown in Figure 1.1. A reliable modern computer vision approach might at best recognize the objects and regions in the image and list the corresponding nouns—road, car, tree and grass. However, when we humans look at the same image, we can not only infer what is happening at that instant but also predict what can happen next. For example, in the same image, we can predict that the car on the bottom right is either going to go straight or turn left at the intersection. Humans' amazing ability to visualize the future is primarily driven by the rich prior knowledge about the visual world.

Endowing computers with this capability—visual prediction—is one of the most fundamental and difficult tasks in computer vision. We believe the task of visual prediction is important for a number of reasons. First, for intelligent agents and systems, prediction is vital for decision making. In order for computers to interact with their environment, simple activity detection is not always sufficient. For successful interactions, robots need to predict the future and plan accordingly. For example, in order to perform assistive activities, robots must be able to predict the intentions of other agents in the scene. Even a task as simple as walking through a crowded hallway requires the prediction of human trajectories. Second, prediction is useful for anomaly detection. Third, prediction requires deep understanding of the visual world and complex interplay between different elements of the scene. Therefore, prediction can act as a way to define full scene understanding. and the task of visual prediction can act as the litmus test for recognition. Finally, predicting the temporal context of a scene or video can serve as a "pretext task" [16] for representation learning—useful for a variety of recognition tasks.

In this thesis, we work toward this goal of generalized visual prediction—determining what is active in the scene as well as how the activity should unfold. However, this leaves us with major questions. What do we predict? What does the output space of visual prediction look like? We consider approaches to prediction that do not rely on human labeling and rely on as little semantic information as possible. Our ultimate goal is to predict features which are freely available from video such as pixels, motion information, or even higher level information. This approach yields many advantages. Unlabeled video data is easy to obtain and is very abundant. In addition, a data-driven approach opens the door for representation learning in images and video.

Unsupervised, data-driven visual prediction is going to rely on a number of important subtasks. The first hurdle is identifying *what* exactly is active in the scene. We need to discover objects and entities that are likely to change and move. Second, we need to understand *how* the scene will change. Based on the context, we want to understand how objects will move and how they will change appearance. Finally, we wish to predict the future *possibilities*—unlike many traditional problems such as object classification, a scene has many possible futures. We need to understand the different ways the future of a scene may unfold and enumerate each of their probabilities.

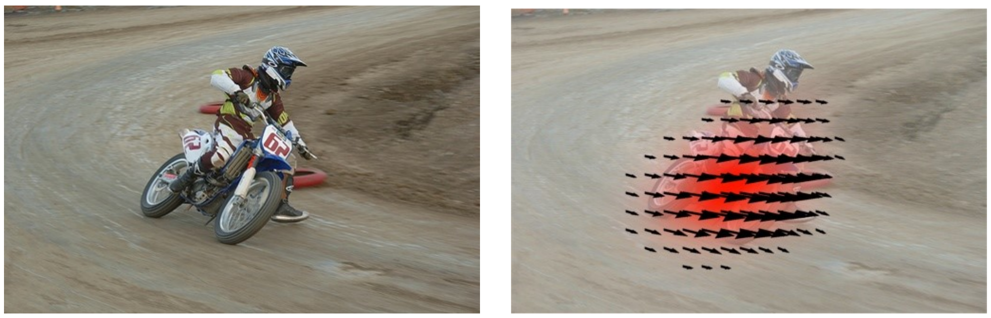


Figure 1.2: Consider this scene of a motorcycle on a dirt track. We can identify the bicycle, man, and the dirt track. However, although this is a static image, we can also infer motion. In Chapter 3 we describe a model that can infer optical flow in static scenes.

This dissertation considers models that forecast events in videos and images in a data-driven, unsupervised way. To this end, we aim to utilize information that is freely available from unlabeled video—pixels, motion vectors, mid-level elements, and more. We apply machine learning models which predict events stochastically, outputting a probability distribution of future events given the scene. We aim towards structured representations for visual forecasting, separating the components which signify the spatial configuration of the scene from direct manipulation of pixels. Our contributions are the following:

1. A model that infers a probability distribution of optical flow over arbitrary scenes.
2. A way to infer the probability density of long-term pixel motion in a scene.
3. An approach, based on Markovian assumptions, that utilizes mid-level elements as a state space for visual prediction over arbitrary timespans.
4. A model that utilizes human pose as structured space for video prediction. This approach uses a recurrent VAE to infer in pose space and a generative adversarial network to infer in pixel space.

Overview

We first consider simple pixel motion as a proxy to direct pixel prediction. Initially we explore the direct prediction of instantaneous motion in static images. We then generalize this idea to longer timeframes with pixel trajectory prediction. Both optical flow and pixel trajectories have been useful for action recognition [26, 99, 115]. For these approaches, we employ convolutional neural networks. CNNs have been extremely successful for visual learning tasks where large amounts of data are available. In this case, existing motion algorithms [115, 122] provide automatically generated labels for our tasks. Using these labels, CNNs are able to learn a representation to discover the active objects in scenes and predict events purely in terms of motion. In order to handle the uncertainty in predicting the future, we explore regression as classification (Chapter 3) as well as variational autoencoders (Chapter 4).

Optical Flow Prediction in Static Scenes: (Chapter 3) Here we present a convolutional neural network based approach for motion prediction. Given a static image, this CNN predicts the immediate future motion of each and every pixel in the image in terms of optical flow. Our model leverages the data in tens of thousands of realistic videos for training. Our method relies on absolutely no human labeling and is able to predict motion based on the context of the scene. Because our model makes no assumptions about the underlying scene, it can predict future optical flow on a diverse set of scenarios. We outperform all previous approaches by large margins.

Forecasting Pixel Trajectories with Variational Autoencoders: (Chapter 4) We then focus on pixel motion over a longer time period—what will move in the scene, where it will travel, and how it will deform over the course of one second. We propose a conditional variational autoencoder as a solution to this problem. In this framework, direct inference from the image shapes the distribution of possible trajectories while latent variables encode information that is not available in the image. We show that our method predicts events in a variety of scenes and can produce multiple different predictions for an ambiguous future.

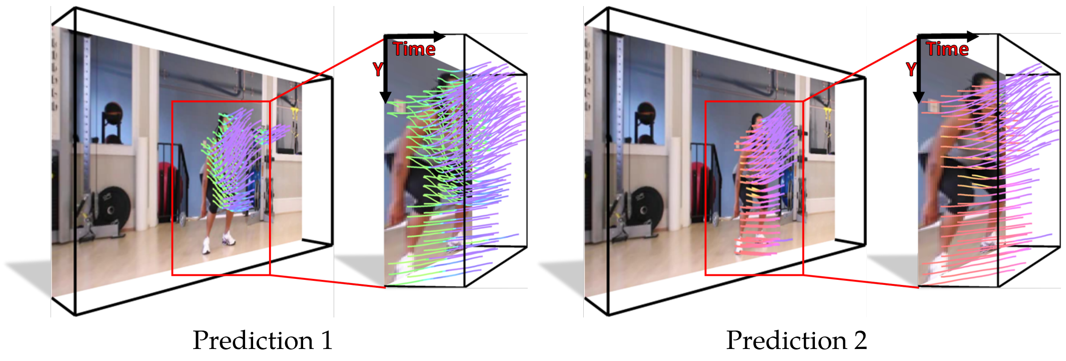


Figure 1.3: Consider this picture of a woman in the gym—she could move up or down. In Chapter 4, we apply variational autoencoders to predict multiple correct one-second motion trajectories given the scene.

We also find that our method learns a representation that is applicable to other tasks.

We find that motion prediction is useful for short timespans. However, quality quickly degrades as the timespan increases. How do we predict events that might happen further in time? On a general level, a state-space approach to forecasting such as [32,43,54], would be applicable. In this case, we consider the image to be an initial "state," and we then manipulate the image to a future visual state. Ideally, we would like our state space to be direct pixels. However, such an approach is non-trivial. Pixel space is very low-level—the dimensionality of the output space is very high, and it is difficult to encode constraints on the output space. For example, pixels can change colors every frame. There is also an averaging effect of multiple possible predictions which leads to blurry predictions. Instead, we could predict events in the space of mid-level visual elements—objects and parts of objects discovered in a data-driven manner. In this way, we are able to predict rough pixel appearance changes as the event unfolds; different mid-level patches may correspond to different perspectives of the same object. Once these visual elements are discovered, the representation, we can train a Markovian model that describes how they may change and move in the scene. We can then leverage this model to estimate the distribution of possible futures.

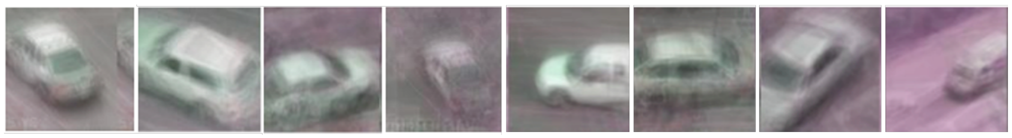


Figure 1.4: Visual elements discovered in an self-supervised manner from Chapter 5. We can use these data-derived elements as an approximation to pixels and predict how they may move or transition into each other over the course of time.

Unsupervised Prediction with Mid-Level Elements: (Chapter 5) In this chapter we move beyond the direct prediction of low-level features; we take an initial step toward forecasting in intermediate visual spaces. We present a conceptually simple but surprisingly powerful method for visual prediction which combines the effectiveness of mid-level visual elements with temporal modeling. Our framework can be learned in a completely unsupervised manner from a large collection of videos. However, more importantly, because our approach models the prediction framework on these mid-level elements, we can not only predict the possible motion in the scene but also predict visual appearances—how are appearances going to change with time. This yields a visual "hallucination" of probable events on top of the scene. We show that our method is able to accurately predict and visualize simple future events; we also show that our approach is comparable to supervised methods for event prediction.

Using Structure for Forecasting: (Chapter 6) In Chapters 3, and 4, we take pixel motion based approaches to data-driven prediction. In doing so we apply generative models (Chapter 4) to the problem of prediction; this approach allows us to handle the ambiguity inherent in prediction the future. Given a scene or video, we can attempt to generate multiple reasonable futures for the scene. However, these low-level motion features are limited by short timescales, visual predictions are still blurry, and they are sometimes difficult to interpret. In Chapter 5, we take an initial step instead towards predicting intermediate features and explore the use of mid-level visual elements as a representation for visual prediction. This initial exploration allowed us to push the temporal boundaries of our forecasting models. In Chapter 6, we build on top of these ideas from previous chapters, using generative

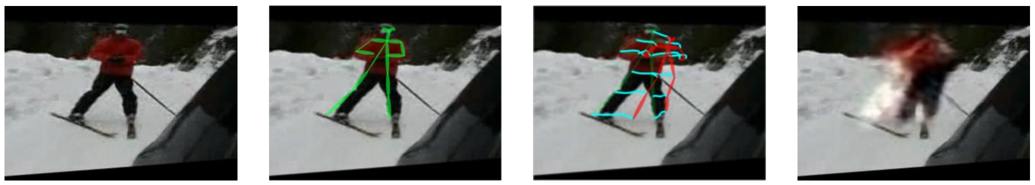

(a) Input Image (b) Detected Pose (c) Predicted Pose (d) Predicted Pixels

Figure 1.5: In Chapter 6, we explore structured spaces as support for video prediction. We use human pose as one such example. Given an initial scene (a), we can detect the human skeleton in (b). We then reason in this high-level, low-dimensional space first to predict an event (c). Given this structure, we can then proceed to low level details such as pixels (d).

models such as variational autoencoders to forecast predictions in a high level, structured space such as pose or semantic boundaries.

We aim to leverage the constraints of structure to forecast events. Approaches that attempt to work directly on pixels often lead to predictions that are blurred beyond the point of interpretability. This is due to a number of reasons. First, many contemporary approaches model the future as deterministic. This leads to blurred predictions as the model outputs a “mean” of all the possible futures in order to minimize prediction error. We can assuage this problem by using stochasticity from a variational autoencoder or an adversarial network. However, even variation autoencoders and generative adversarial networks, despite stochasticity, struggle to predict interpretable results when direct pixel prediction is considered. In this chapter, we propose to train a generative model of future video not on direct pixels but on higher-level, structured output spaces which are still visually meaningful. We demonstrate the use of human pose as possibility. Other choices include motion segments, semantic masks, and camera parameters. We can still leverage stochasticity in VAEs and GANs with this approach. However, these output spaces are inherently more tractable than direct pixels.

Chapter 2

Related Work

Prediction—temporal forecasting—is a major component of intelligence [40]. Researchers in neuroscience have found extensive support for sensory prediction in the human brain [4], and others have even found evidence in animals such as rats and pigeons [130]. In the vision community, forecasting has caught interest in recent years with researchers focusing on separate aspects of the problem. The first aspect is what should be predicted—what is the output space of prediction. One could consider the very basic elements of video—pixels and motion. Some initial work attempted to take forecasting to the lowest level and model direct pixels. However, these approaches are problematic; direct modeling of low-level features is very difficult with current machine learning approaches. Many of these papers show results on visually limited datasets such as handwritten characters or shapes, but they fail to cope with datasets that are visually diverse. Others have considered more semantic forms of prediction—predicting the action class of what is going to happen next. Semantic prediction also has drawbacks however; it tells us nothing about the future action beyond the category—direct spatio-temporal information concerning objects is often lost. On the contrary, this thesis strives to go beyond semantic classification and predict the spatial layout of future actions.

2.1 Semantic Prediction

The first main approach to visual forecasting is a semantic, modeling based approach. Here we make explicit, human-chosen assumptions of what entities should be forecast in the scene. Semantic prediction can be further subdivided into two categories—agent-based prediction and early event detection.

2.1.1 Agent-based Prediction

In agent based prediction, we make assumptions on what are the active elements in the scene. Agents may be cars, people, or robots. Once this assumption is made, we develop an explicit model to predict agent behavior. Tracking based approaches focus on modeling agent behavior at short time scales and hence use linear models [50]. For longer term predictions, models such as Markov Decision Process (MDP) [54, 134], Markov Logic Networks [105], ATCRF [56], CRF [30], and AND-OR graphs [39,86] are used. Pedestrian trajectory forecasting specifically has been a major focus in this area. Approaches to model future pedestrian trajectories include Inverse Reinforcement Learning [54, 94], Social Forces [2], Convolutional Neural Networks [126], and Bayesian Networks [3]. Modeling the future trajectories of cars [64, 127] has also been a prominent application domain. Numerous papers have explicitly focused on modeling the dynamics of human skeletons in the visual world [9, 32, 47] Emerging variations on agent-based forecasting include multi-agent scenarios [2, 43, 75], human-object interations through graphical models [30, 56], ego-centric video [82, 94], and sports domains [27, 103]. The work described in Chapter 5 may be considered agent-centric; however, in our case, the agents are data-driven and not explicitly semantic. In general, while agent-based models are often interpretable and powerful, they often rely on strongly supervised data.

2.1.2 Early Event Detection

An alternative approach to semantic prediction is to simply treat the problem as a form of action detection. Instead of modeling the spatial configuation of a pre-chosen agent, we wish to detect and classify future video events into pre-chosen action classes. Pioneers of this approach include Ryoo et al. [95,96] with bag of words in video and Hoai et al. [42] with SOSVMs for facial expression prediction. Others have taken early event detection to more general settings using hierarchical representations [62], LSTMs [74], feature learning [55], and inverse reinforcement learning [129] to infer future action classes in third person video as well as ego-centric [132]. Some have taken semantic abstraction without explicit classes, inferring future high level semantic features from convolutional neural networks [110].

Predicting only semantic information has multiple drawbacks. First, such approaches make strong assumptions on the domain—humans choose what is going to temporally change. Furthermore, they require extensive human labeling which is difficult to acquire. In addition, action classes tell us nothing about the geometry of the scene or how the spatial configuration of objects may change over time. On the contrary, a data-driven approach—relying on as few assumptions on the data as possible—gives us an ability to train our prediction framework in a completely self-supervised manner. With rich visual representations we can forecast visual appearances and geometry as well. Finally, a self-supervised approach is a form of temporal context prediction, allowing unsupervised representation learning from video.

2.2 Low Level Prediction

Because of the limitations of semantic information, many papers have focused on a low level approach. These approaches shun any assumption concerning active agents or the environment. Instead, they rely on forecasting low level information such as pixels or motion by leveraging large video databases. These approaches can be at best be divided into two subcategories. The first is an approach based on predicting only pixel motion. This is advantageous as the model only needs to focus on predicting the motion of an object and not the intermediate pixel appearances. The other category is direct pixel prediction.

2.2.1 Motion Prediction

Inferring pixel motion in static imagery has been an active research problem for almost a decade. For instance, Yuen et al. [128] is an early work which utilizes a nearest-neighbor approach based on scene matching. In this paper, the proposed approach retrieves videos similar to the input scene and then builds a model of expected motion given the retrievals. However, since the matching is done based on the scene, this requires extraordinarily large amounts of training data; one needs to represent all possible spatial and temporal configurations of objects in the world explicitly. Therefore, warping [67] is needed to generate predictions in case the retrievals are close but not identical. The work of Pintea et al. [87] uses a more complex model, namely a structured random forest for motion prediction on the KTH [63] dataset. Furthermore, Brabandere et al. [13] propose transformative filters, showing compelling results on moving MNIST characters. In the context of robotics, Finn et al. [28] show an effective LSTM-based approach to motion prediction on videos.

Unfortunately, these approaches easily break down when exposed to more realistic datasets. We find empirically in Chapter 3 that both the SRF approach [87] and the nearest neighbor approach [128] seem to struggle with more realistic datasets such as the UCF-101 [101]. In this thesis we explore CNN-based approaches to motion prediction that are able to handle visual data that is highly varied. Furthermore, we explore generative models such as variational autoencoders that address the ambiguity inherent in motion prediction. In all cases we take a self-supervised approach to training our models.

2.2.2 Pixel Prediction

Some researchers have taken the most direct approach—predicting raw pixels in video frames. Such an approach relies on no assumptions, forecasts all information, and is visually interpretable. Initial work has focused on combining CNNs with recurrent neural network models. For example, Ranzato et al. [90] use a LSTM to predict the immediate next frame given a video input. Similarly, Srivastava et al. [102] use a recurrent sequential encoder-decoder architecture to predict motions of handwritten characters from a video.

Patraucean et al. [85] extend this LSTM recurrent approach even further with convolutional LSTM units. In contrast to recurrent approaches, Lotter et al. [69] propose predictive coding networks for first-person car videos. However, even deep networks still struggle with underfitting with more complex datasets. These models often lead to blurring for two reasons. First, many of these models are deterministic and do not account for the uncertain nature of future evens. For instance, the model in [102] could only predict one future for a given input. This forces the model to average over all possible futures. Second, many of these models [85, 90, 102] attempt to predict frames at a very fine timescale, specifically one frame at a time at a rate of thirty frames a second. In this case, the model must predict every fine aspect of motion sequentially in addition to pixel appearance. Recently there have been attempts to train generative models with popular approaches such as adversarial networks, variational autoencoders, and pixelCNN architectures. The authors in [125] train a VAE that predicts pixels for future frames. They apply their approach to images of video game sprites, moving shapes, and humans against a white background. Kalchbrenner et al. [49] propose a model that estimates the joint distribution of pixels in a video. They test their architecture on moving MNIST characters and a constrained robotic environment. Others [71, 76, 111, 112, 133] have attempted to use generative adversarial networks for video prediction. These methods seem the most promising for unconstrained, realistic videos. However, they still struggle with blurriness and outputs that are even recognizable.

Most recently some papers have proposed a combined approach of motion and pixel modeling [66]. However, we argue in this thesis that reasoning on the underlying structure in an image is a promising approach for forecasting. For instance, in Chapter 5, we build on mid-level discriminative patch discovery [18, 20, 25, 100] to create a dictionary of visual elements for prediction. Because the basic elements in our framework are based on these mid-level patches, this approach does not need to directly model the fine changes in pixels or motion. At the same time, the mid-level elements allow us to model rough changes in pixel appearances. In Chapter 6, we show another hierarchical approach using human pose. We first reason about the future purely in human pose space and then leverage this structure to fill in the low-level details.

2.3 Generative Models

Generative models of the visual world—both for images and video—constitute another active area of research. Fundamentally, the assumption is that the visual world may be modeled a series of stochastic variables; for example, some variables may represent the type of objects, the size of them, their position, and more. If we are able to discover these stochastic variables, we may sample from them to generate novel images. Variational autoencoders have already shown promise in a number of domains involving generating pixels, including handwritten digits [53,98], faces [53,93], house numbers [52], CIFAR images [36], and even face pose [60]. Furthermore, adversarial networks [15,34,46,84,89,119], have shown promise as well, generating almost photo-realistic images for particular datasets. However, it seems that adversarial networks seem to work the best when given additional structure [46, 84]. There is also a third line of work regarding pixelCNNs and pixelRNNs [38, 106, 107] With these methods, the joint distribution of pixels is modeled directly. In Chapter 5, we apply VAEs to generate multiple possible futures given an input image or video. That is, we assume that the future is non-deterministic and can be modeled by latent variables sampled from a particular—usually Gaussian—distribution. We can then sample these latent variables to visualize multiple possibilities. In Chapter 6, we train a stochastic generative model over a higher level, structured space and then exploit this structure to aid the pixel prediction of a GAN. In our case use specifically use the structure of human pose, but other potential approaches include semantic segments or even camera parameters. Recent work suggests that PixelCNNs could also benefit from structure [92] and generate results on the pixel level.

2.4 Concurrent Work and Emerging Trends

In this thesis we argue that structure is useful for data-driven video forecasting—self-supervised approaches which rely on as few assumptions on the data as possible. Instead of attempting to model pixels directly, an approach which first reasons on a higher-level factor of variation—mid-level patches, human pose, segments—is going to be far more tractable.

Concurrent work by others has reinforced this position, showing the use of forecasting in semantic segment space [72] and human pose down to the pixel level [109]. Recent trends are moving from passive prediction—focused in this thesis—to more interactive prediction. Assuming an agent that can choose actions in the visual world, how will these actions affect the future state? Multiple papers in both the context of robotics [7, 28] and reinforcement learning [22,79,83,108] have begun to explore interactive forecasting. Finally, there has been the topic of intuitive physics. Can we get computers to forecast the physical dynamics of objects in the visual world? Many recent papers [31,77,78,124] are initial explorations.

Chapter 3

Optical Flow Prediction in Static Scenes

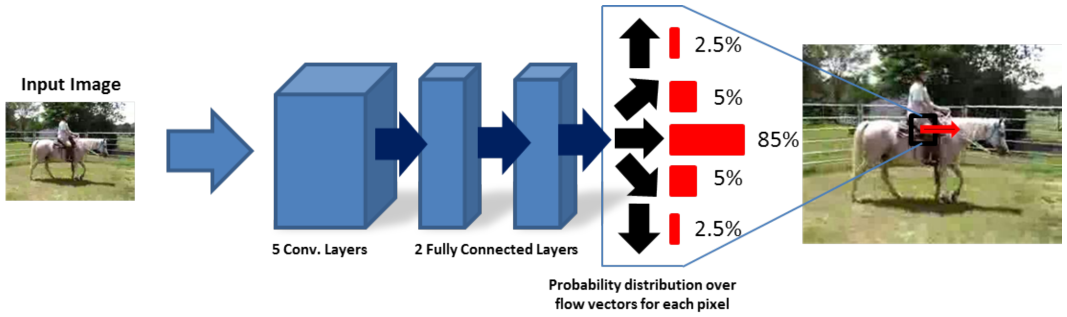


Figure 3.1: **Overview**. Our network is similar to the standard 7-layer architecture [58] used for many recognition tasks. We take a 200x200 image as input. However, we use a spatial softmax as the final output. For every pixel in the image we predict a distribution of various motions with various directions and magnitudes. We can combine a weighted average of these vectors to produce the final output for each pixel. For computational reasons, we predict a coarse 20x20 output.

3.1 Introduction

In this chapter, we take an initial step towards generalized prediction. Specifically, we look at the task of motion prediction—given a static image we predict the dense expected optical flow as if this image was part of a video. This framework can be learned from tens of thousands of realistic videos. The framework can work in indoor and outdoor environments if

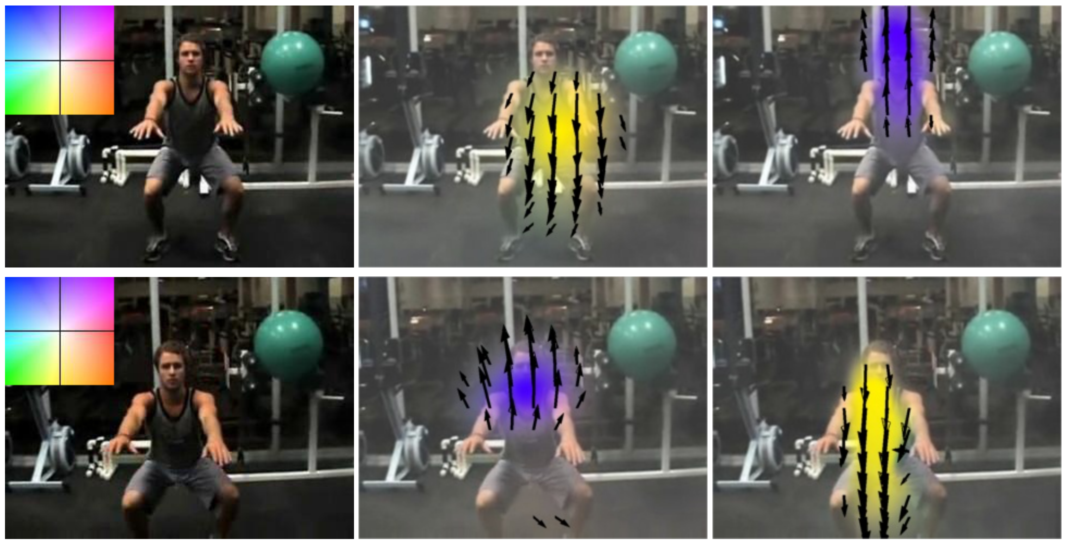

(a) Input Image (b) Prediction (c) Ground Truth

Figure 3.2: Consider the images on the left. Is the man squatting up or down? The bottom is near completion (or just starting), and the top image is right in the middle of the action. Our dataset contains a large number of ambiguous images such as these.

the agent is an animal, a human, or even a car. It can account for one or multiple agents. Of course, we can see that motion prediction is a highly-context dependent problem. The future motion not only depends on what is active in the scene but also its context. For example, someone's entire body may move up or down if they are jump-roping, but most of the body will be stationary if they are playing the flute. Instead of modeling agents and its context separately under restrictive assumptions, we use a learning based approach for motion prediction. Specifically, we train a deep network that can incorporate all of this contextual information to make accurate predictions of future motion in a wide variety of scenes. We train our model from thousands of realistic video datasets, namely the UCF-101 [101] and the HMDB-51 [59].

In this chapter, we make three contributions. First, we present a CNN model for motion prediction. Given a static image, our CNN model predicts expected motion in terms of optical flow. Our CNN-based model is agent-free and makes almost no assumptions about the underlying scene. Therefore, we show experimental results on diverse set of scenes. Second, our CNN model gives state of the art performance on prediction compared to contemporary approaches. Finally, we also present a proof of concept extension of CNN model which makes long-range prediction about future motion. Our preliminary results indicate that this new CNN model might indeed be promising even on the task of long-range prediction.

3.2 Methods

3.2.1 Regression as Classification

Intuitively, motion estimation can be posed as a regression problem since the space is continuous. Indeed, this is exactly the approach used in Pintea et al. [87], where the authors use structured random forests to regress the magnitude and direction of the optical flow. However, such an approach tends to smoothen results. The approach handles ambiguity by averaging out the flow. Interestingly, in a related problem of surface normal prediction, researchers have proposed reformulating structured regression as a classification problem [61, 118]. Specifically, they quantize the surface normal vectors into a codebook of clusters, and then the output space becomes predicting the cluster membership. In our work, we take a similar approach. We quantize optical flow vectors into 40 clusters by k-means. We can then treat the problem in a manner similar to semantic segmentation where we classify each region as the image as a particular cluster of optical flow. We use a soft-max loss layer at the output for computing gradients.

However, at test time, we create a soft output by considering the underlying distribution of all the clusters, taking a weighted-probability sum over all the classes in a given pixel for the final output. Transforming the problem into classification also leads directly to a discrete probability distribution over vector directions and magnitudes. As the problem of motion prediction can be ambiguous depending on the image (see Figure 3.2), we can utilize this probability distribution over directions to measure how informative our predictions are. We may be unsure if the man in Figure 3.2 is sitting down or standing up given only the image, but we can be quite sure he will not turn right or left. In the same way, our network can rank upward and downward facing clusters much higher than other directions.

Even if the ground truth is upward, and the highest ranked cluster is downward, it may be that the second-highest cluster is also upward. A discrete probability distribution, through classification, allows an easier understanding of how well our network may be performing.

3.2.2 Network Design

Our model is similar to the standard seven-layer architecture from [58]. To simplify the description, we denote the convolutional layer as $C(k, s)$, which indicates the there are k kernels, each having the size of $s \times s$. During convolution, we set all the strides to 1 except for the first layer, which is 4. We also denote the local response normalization layer as LRN, and the max-pooling layer as MP. The stride for pooling is 2 and we set the pooling operator size as 3×3. Finally, $F(n)$ denotes fully connected layer with n neurons.

Our network architecture can be described as: $C(96, 11) \rightarrow LRN \rightarrow P \rightarrow C(256, 5) \rightarrow LRN \rightarrow P \rightarrow C(384, 3) \rightarrow C(384, 3) \rightarrow C(256, 3) \rightarrow P \rightarrow F(4096) \rightarrow F(4096)$. We used a modified version of the popular Caffe toolbox [48] for our implementation. For computational simplicity, we use 200x200 windows as input. We used a learning rate of 0.0001 and a stepsize of 50000 iterations. Other network parameters were set to default. The only exception is that we used Xavier initialization of parameters. Instead of using the default softmax output, we used a spatial softmax loss function from [118] to classify every region in the image. This leads to a $M \times N \times K$ softmax layer, where M is the number of rows, N is the number of columns, and C is the number of clusters in our codebook. We used $M = 20$, $N = 20$, and $K = 40$ for a softmax layer of 16,000 neurons. Our softmax loss is spatial, summing over all the individual region losses. Let I represent the image and Y be the ground truth optical flow labels represented as quantized clusters. Then our spatial loss function $L(I, Y)$ is the following:

$$L(I, Y) = -\sum_{i=1}^{M \times N} \sum_{r=1}^{K} (\mathbb{1}(y_i = r) \log F_{i,r}(I) \tag{3.1}$$

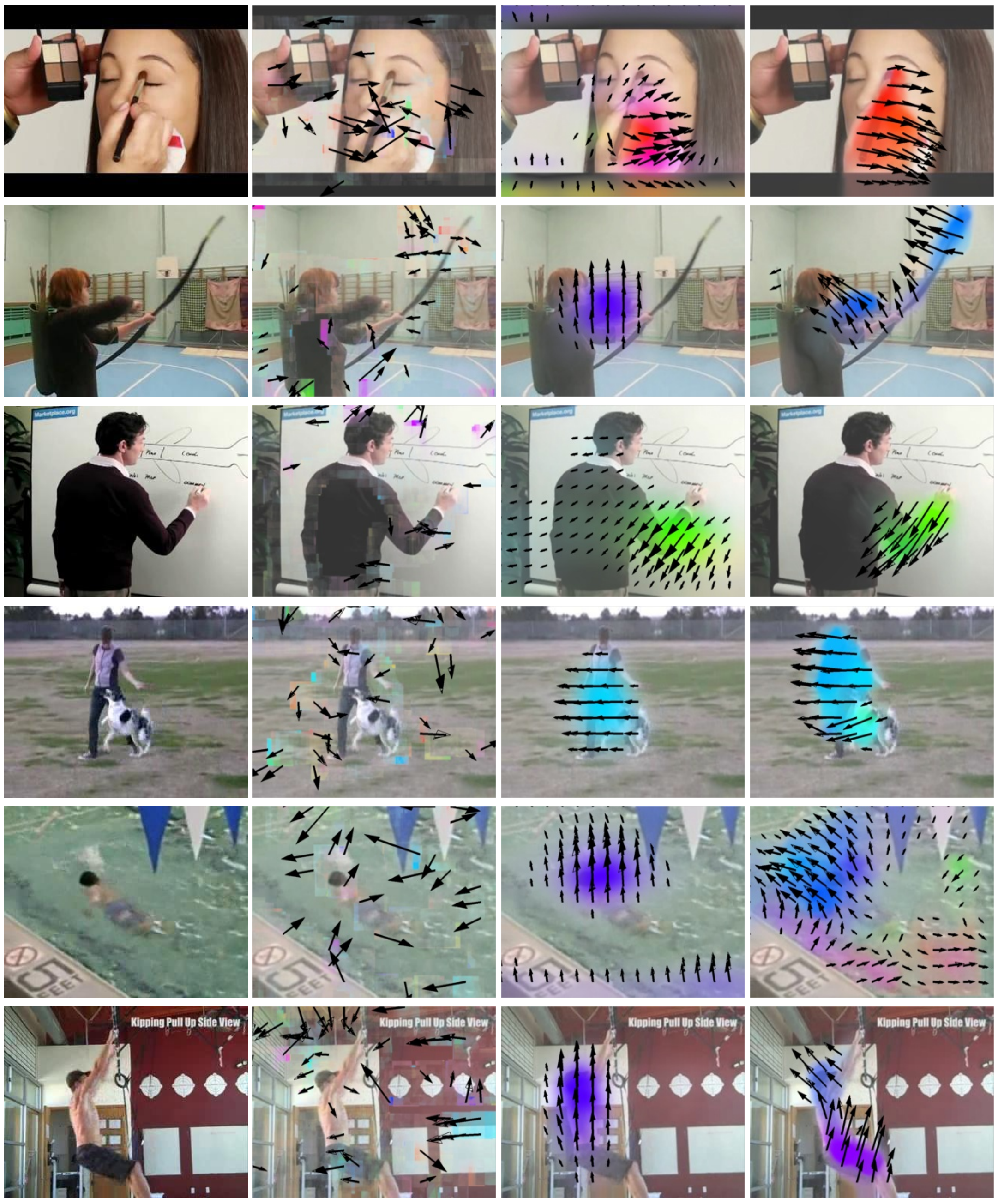
$F_{i,r}(I)$ represents the probability that the ith pixel will move according to cluster r. $\mathbb{1}(y_i = r)$ is an indicator function.

Data Augmentation: For many deep networks, datasets which are insufficiently diverse or too small will lead directly to overfitting. Simonyan et al. [99] and Karpathy et al. [51] show that training directly on datasets such as the UCF-101 for action classification leads to overfitting. There are only data on the order of tens of thousands of videos. However, our problem of single-frame prediction is different from this task. We find that we are able to build a generalizable representation for prediction by training our model over 350,000 frames from the UCF-101 dataset as well as over 150,000 frames from the HMDB-51 dataset. We benefit additionally from data augmentation techniques. We randomly flip each image as well as use randomly cropped windows. For each input, we also mirror or flip the respective labels. In this way we are able to avoid spatial biases— such as humans always appearing in the middle of the image—and train a general model on a far smaller set of videos than for recognition tasks.

Labeling: We automatically label our training dataset with an optical flow algorithm. With a publically available implementation, we chose DeepFlow [122] to compute optical flow. The UCF-101 and the HMDB-51 dataset use realistic, sometimes low-quality videos from a wide variety of sources. They often suffer from compression artifacts. Thus, we aim to make our labels somewhat less noisy by taking the average optical flow of five future frames for each image. The videos in these datasets are also unstabilized. Wang et al. [115] show that action recognition can be greatly improved with camera stabilization. In order to further denoise our labels, we wish to focus on the motion of objects inside the image, not the camera motion. We thus use the stabilization portion of the implementation of Wang et al. [115] to automatically stabilize videos using an estimated homography.

3.3 Experiments

For our experiments, we mostly focused on two datasets, the UCF-101 and HMDB-51 which have been popular for action recognition. For both of these datasets, we compared against baselines using 3-fold cross validation with the splits specified by the dataset organizers. We also evaluated our method on the KTH 3.6 dataset using the exact same configuration



(a) Input (b) [87] (c) Ours (d) GT

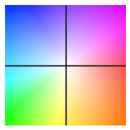
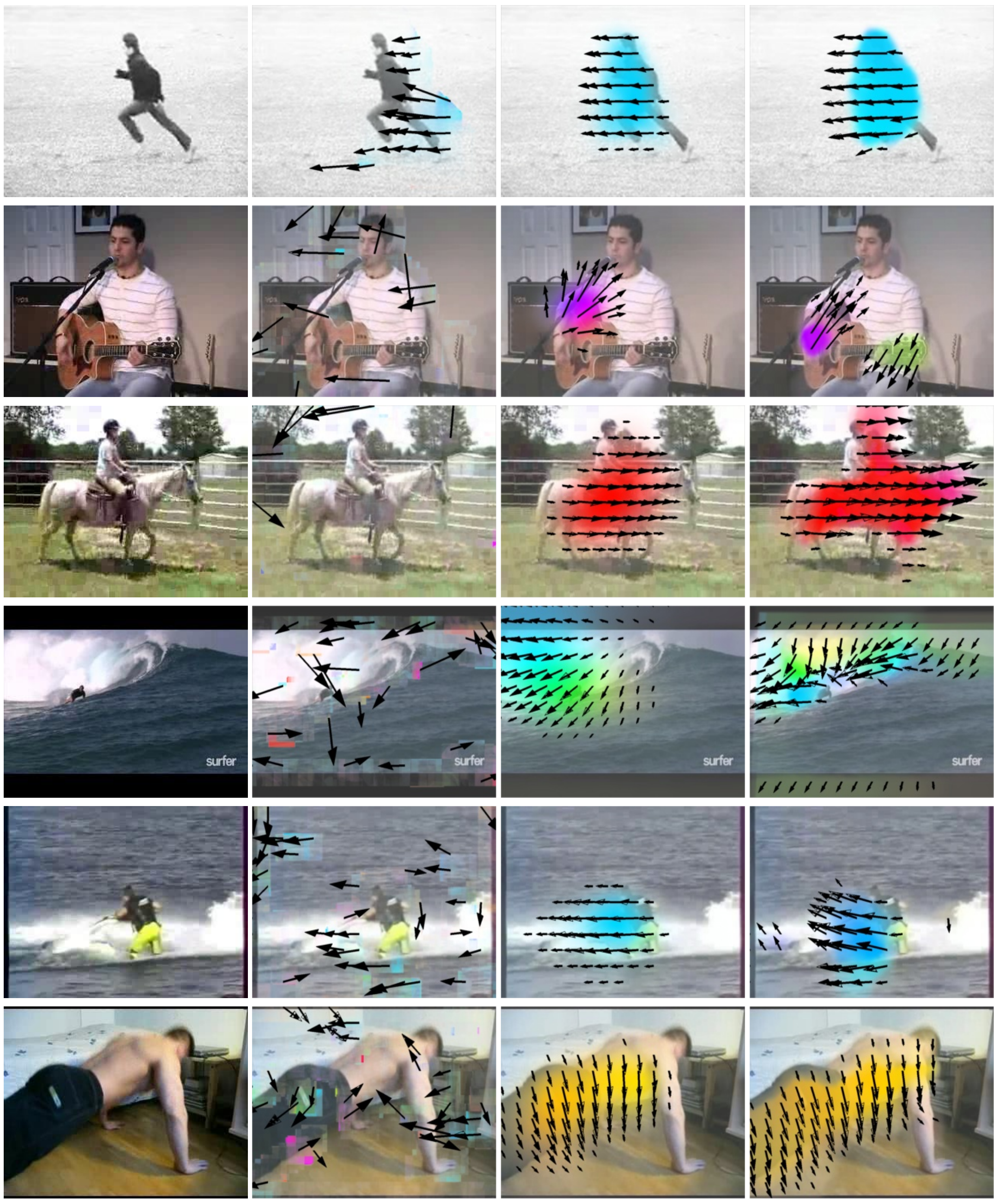
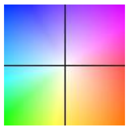


Figure 3.3: Qualitative results from our method for the single frame model. While [87] is able to predict motion in the KTH dataset, we find our network strongly outperforms the baseline on more complex datasets. Our network can find the active elements in the scene and correctly predict future motion based on the context in a wide variety and scenes and actions. The color coding is on the right.



(a) Input (b) [87] (c) Ours (d) GT

Figure 3.4: Qualitative results from our method for the single frame model. While [87] is able to predict motion in the KTH dataset (top left), we find our network strongly outperforms the baseline on more complex datasets. Our network can find the active elements in the scene and correctly predict future motion based on the context in a wide variety and scenes and actions. The color coding is on the right.



UCF-101

Method	EPE	EPE-Canny	EPE-NZ
SRF [87]	1.30	1.23	3.24
NN pooled-5	2.31	2.20	4.40
NN fc7	2.24	2.16	4.27
Ours-HMDB	1.35	1.26	3.26
Ours	**1.27**	**1.17**	**3.19**

—	Dir	Dir-Canny	Dir-NZ
SRF [87]	.004	.000	-.013
NN pooled-5	-.001	-.001	-.067
NN fc7	-.005	-.006	-.060
Ours-HMDB	0.017	0.007	0.032
Ours	**.045**	**.025**	**.092**

—	Orient	Orient-Canny	Orient-NZ
SRF [87]	.492	.600	.515
NN pooled-5	.650	.650	.677
NN fc7	.649	.649	.651
Ours-HMDB	.653	.653	.672
Ours	**.659**	**.657**	**.688**

—	Top-5	Top-5-Canny	Top-5-NZ
SRF [87]	79.4%	81.7%	10.0%
NN pooled-5	77.8%	79.5%	20.0%
NN fc7	78.3%	79.9%	18.8%
Ours-HMDB	88.7%	90.0%	60.6%
Ours	**89.7%**	**90.5%**	**65.0%**

—	Top-10	Top-10-Canny	Top-10-NZ
SRF [87]	82.2%	84.4%	17.2%
NN pooled-5	83.2%	85.3%	32.9%
NN fc7	84.0%	85.4%	32.3%
Ours-HMDB	95.6%	95.9%	88.8%
Ours	**96.5%**	**96.7%**	**90.9%**

(a)

HMDB-51

Method	EPE	EPE-Canny	EPE-NZ
SRF [87]	1.23	1.20	3.46
NN pooled-5	2.51	2.49	4.89
NN fc7	2.43	2.43	4.69
Ours-UCF	1.30	1.26	3.49
Ours	**1.21**	**1.17**	**3.45**

—	Dir	Dir-Canny	Dir-NZ
SRF [87]	.000	.000	-.010
NN pooled-5	-.008	-.007	-.061
NN fc7	-.007	-.005	-.061
Ours-UCF	.016	.011	.003
Ours	**.016**	**.012**	**.030**

—	Orient	Orient-Canny	Orient-NZ
SRF [87]	.461	.557	.495
NN pooled-5	.631	.631	.644
NN fc7	.630	.631	.655
Ours-UCF	.634	.634	.664
Ours	**.636**	**.636**	**.667**

—	Top-5	Top-5-Canny	Top-5-NZ
SRF [87]	81.9%	83.6%	13.5%
NN pooled-5	76.3%	77.8%	14.0%
NN fc7	77.3%	78.7%	13.5%
Ours-UCF	89.4%	89.9%	60.8%
Ours	**90.2%**	**90.5%**	**61.0%**

—	Top-10	Top-10-Canny	Top-10-NZ
SRF [87]	84.4%	86.1%	22.1%
NN pooled-5	82.9%	84.0%	23.9%
NN fc7	83.6%	84.4%	23.2%
Ours-UCF	95.8%	95.9%	87.6%
Ours	**95.9%**	**95.9%**	**87.5%**

(b)

Table 3.1: Single-image evaluation using the 3-fold split on UCF-101. Ours-HMDB represents our network trained only on HMDB data. The Canny suffix represents pixels on the Canny edges, and the NZ suffix represents moving pixels according to the ground-truth. NN represents a nearest-neighbor approach. Dir and Orient represent direction and orientation metrics respectively. For EPE, less is better, and for other metrics, higher is better.

ImageNet-Pretrained vs. Trained from Scratch

Method	EPE	EPE-Canny	EPE-NZ
Pretrained	1.19	1.12	3.12
From Scratch	1.28	1.21	3.21

—	Orient	Orient-Canny	Orient-NZ
Pretrained	.661	.659	.692
From Scratch	.659	.658	.691

—	Top-5	Top-5-Canny	Top-5-NZ
Pretrained	91.0%	91.1%	65.8%
From Scratch	89.9%	90.3%	65.1%

(a)

Stabilization

Method	EPE	EPE-Canny	EPE-NZ
Unstabilized	1.35	1.28	3.60
Stabilized	1.49	1.42	3.61

—	Orient	Orient-Canny	Orient-NZ
Unstabilized	.641	.641	0.664
Stabilized	.652	.652	0.698

—	Top-5	Top-5-Canny	Top-5-NZ
Unstabilized	88.9%	89.2%	63.4%
Stabilized	88.3%	88.8%	59.7%

(b)

Figure 3.5: Here we compare our unsupervised network, trained from scratch, to the same network fine-tuned from supervised AlexNet features on UCF-101. The Canny suffix represents pixels on the Canny edges, and the NZ suffix represents moving pixels according to the ground-truth. Dir and Orient represent direction and orientation metrics respectively. For EPE, less is better, and for other metrics, higher is better.

in Pintea et al. [87] with DeepFlow. Because the KTH dataset is very small for a CNN, we finetuned our UCF-101 trained network on the training data. For training, we subsampled frames by a factor of 5. For testing, we sampled 26,000 frames per split. For our comparison with AlexNet finetuning, we used a split which incorporated a larger portion of the training data. This split includes all but group 5 as training data. We used three baselines for evaluation. First we used the technique of Pintea et al. [87], a SRF approach to motion prediction. We took their publicly available implementation and trained a model according to their default parameters. Because of the much larger size of our datasets, we had to sample SIFT-patches less densely. We also use a nearest-neighbor baseline using both fc7 features from the pre-trained AlexNet network as well as pooled-5 features. Finally, we compare unsupervised training from scratch with finetuning on the supervised AlexNet network.

3.3.1 Evaluation Metrics

Because of the complexity and sometimes high level of label ambiguity in motion prediction, we use a variety of metrics to evaluate our method and baselines. Following from Pintea et al. [87], we use traditional End-Point-Error, measuring the Euclidean distance of the estimated optical flow vector from the ground truth vector.

In addition, given vectors $\mathbf{x}_1$ and $\mathbf{x}_2$, we also measure direction similarity using the cosine similarity distance: $\frac{\mathbf{x}_1^T \mathbf{x}_2}{\|\mathbf{x}_1\| \|\mathbf{x}_2\|}$ and orientation similarity (angle taken on half-circle): $\frac{|\mathbf{x}_1^T \mathbf{x}_2|}{\|\mathbf{x}_1\| \|\mathbf{x}_2\|}$

The orientation similarity measures how parallel is predicted optical flow vector with respect to given ground truth optical flow vector. Some motions may be strictly left-right or up-down, but the exact direction may be ambiguous. This measure accounts for this situation. We choose these metrics established by earlier work. However, we also add some additional metrics to account for the level of ambiguity in many of the test images. As Pintea et al. [87] note, EPE is a poor metric in the case where motion is small and may reasonably proceed in more than one possible direction. We thus additionally look at the underlying distribution of the predicted classes to understand how well the algorithm accounts for this ambiguity. For instance, if we are shown an image as in Figure 3.2, it is unknown if the man will move up or down. It is certainly the case, however, that he will not move right or left. Given the probability distribution over the quantized flow clusters, we check to see if the ground truth is within the top probable clusters. For the implementation of Pintea et al. [87], we create an estimated probability distribution by quantizing the regression output from all the trees and then, for each pixel, we bin count the clusters over the trees. For Nearest-Neighbor we take the top-N matched frames and use the matched clusters in each pixel as our top-N ranking. We evaluate over the mean rank of all pixels in the image.

KTH (DeepFlow)

Method	EPE	EPE-Canny	EPE-NZ
[87]	0.21	0.19	1.72
Ours	**0.19**	**0.18**	**1.17**

—	Orient	Orient-Canny	Orient-NZ
[87]	.30	.32	.75
Ours	**.67**	**.67**	**.90**

—	Top-5	Top-5-Canny	Top-5-NZ
[87]	93.9%	94.4%	2.3%
Ours	**99.0%**	**99.0%**	**98.0%**

Figure 3.6: We compare our network fine-tuned on the KTH dataset to [87]. For EPE, less is better, and for other metrics, higher is better. Top-5-NZ is very low for [87] as it often correctly predicted flow orientation but at a low magnitude. Orient represents orientation metric. NZ and Canny are non-zero and Canny pixels.

Following Pintea et al. [87], we also evaluate over the Canny edges. Because of the simplicity of the datasets in [87], Canny edges were a good approximation for measuring the error of pixels of moving objects in the scene. However, our data includes highly cluttered scenes that incorporate multiple non-moving objects. In addition, we find that our network is very effective at identifying moving vs non-moving elements in the scene. We find that the difference between overall pixel mean and Canny edges is very small across all metrics and baselines.Thus, we also evaluate over the moving pixels according to the ground-truth.

Moving pixels in this case includes all clusters in our codebook except for the vector of smallest magnitude. While unfortunately this metric depends on the choice of codebook, we find that the greatest variation in performance and ambiguity lies in predicting the direction and magnitude of the active elements in the scene.

3.3.2 Qualitative Results

Figure 3.4 shows some of our qualitative results. For single frame prediction, our network is able to predict motion in many different contexts. We find that while the SRF approach is able to make reasonable predictions on the KTH, qualitative performance collapses once the complexity and size of the dataset increases. Although most of our datasets consist of human actions, our model can generalize beyond simply detecting general motion on humans. Our method is able to successfully predict the falling of the ocean wave in the second row, and it predicts the motion of the entire horse in the first row. Furthermore, our network can specify motion depending on the action being performed. For the man playing guitar and the man writing on the wall, the arm is the most salient part to be moved. For the man walking the dog and the man doing a pushup, the entire body will move according to the action.

3.3.3 Quantitative Results

We show in Table 3.1 that our method strongly outperforms both the nearest-neighbor and SRF-based baselines by a large margin by most metrics. This holds true for both datasets. Interestingly, the SRF-based approach seems to come close to ours based on End-Point-Error on all datasets, but is heavily outperformed on all other metrics. This is largely a product of the End-Point-Error metric, as we find that the SRF tends to output the mean—optical flow with very small magnitude. This is consistent with the results found in Pintea et al. [87], where actions with low, bidirectional motion can result in higher EPE than predicting no motion at all. When we account for this ambiguity in motion in the top-N metric, however, the difference in performance is large. For KTH, the SRF approach is close in EPE and Orientation, but Top-N suffers greatly because it often output vectors of correct direction but incorrect magnitude. Our method is able to generalize beyond the structure of a particular dataset, as testing on one dataset and training on another leads only to a small loss in performance. Finally, in Table 3.5 we find even without camera stabilization that the difference in performance is small. Compared to unsupervised training from scratch, we find that finetuning from supervised, pretrained features yield only a very small improvement in performance. Looking over all pixels, the difference in performance between approaches is small. On absolute levels, the orientation and top-N metrics also tend to be high. This is due to the fact that most pixels in the image are not going to move. Outputting low or zero-motion over the entire image can thus lead to good performance for many metrics. Canny edge pixels yield similar results, as our natural images often include background clutter with objects that do not move. The most dramatic differences appear over the non-zero pixels. The direction metric is for our method is low at $.09$ because of direction ambiguity, but orientation similarity is much larger. The largest performance gains come at the top-N rankings. For 40 clusters, random chance for top-5 is 12.5%, and for top-10 it is 25%. Nearest-neighbor does slightly better than chance, but SRF actually performs slightly worse. This is most likely due to the SRF tendency to output low magnitude flow. Our method performs much better, with the ground truth direction and magnitude vector coming in 65% of the time in the top-5 ranking, and to a very high 90.9% of the time in the top ten.

3.4 Multi-Frame Prediction

Until now we have described an architecture for predicting optical flow given a static image as input. However, it would be interesting to predict not just the next frame but a few

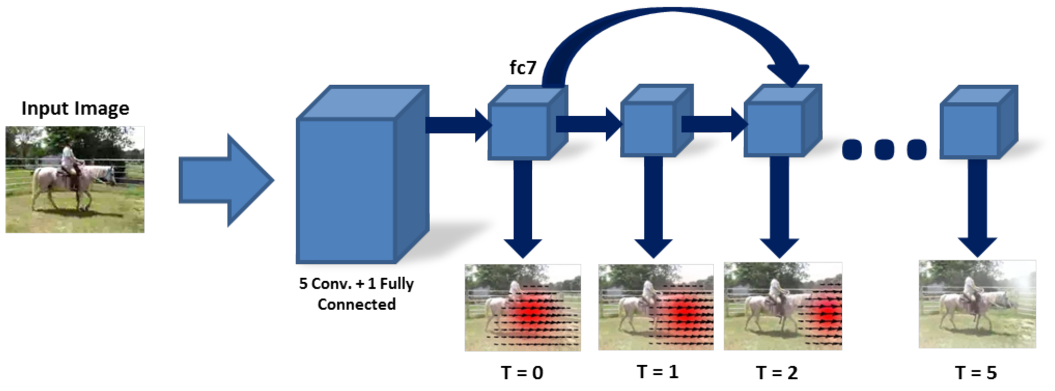
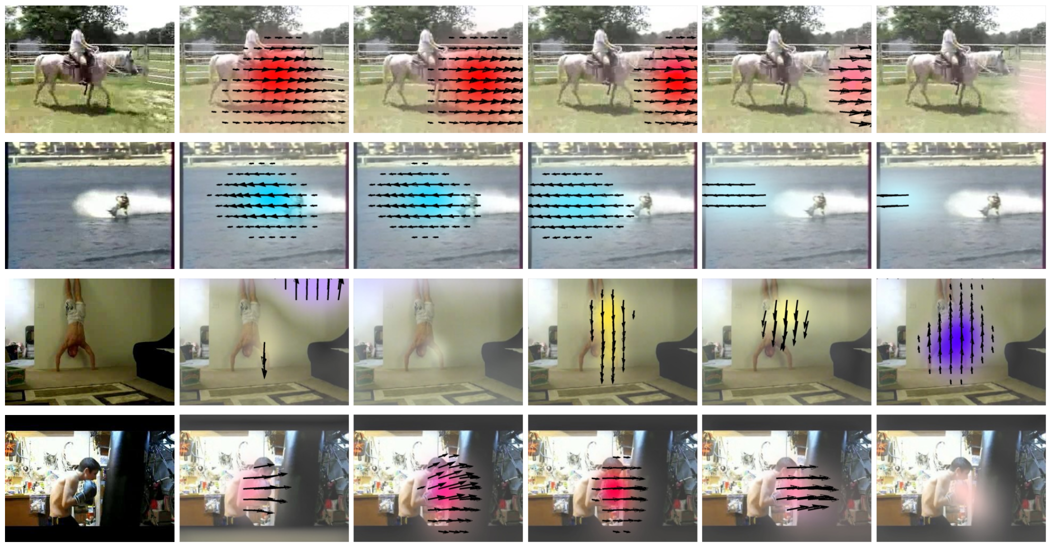


Figure 3.7: **Overview**. For our multiframe prediction, we predict entire clustered frames of optical flow as a sequence of frames. We take the learned features for our single frame model as our input, and we input them to a series of six fully connected layers, with each layer having access to the states of the past layers.

seconds into future. How should we design such a network?

We present a proof-of-concept network to predict 6 future frames. In order to predict multiple frames into the future, we take our pre-trained single frame network and output the seventh feature layer into a "temporally deep" network using the implementation of Donahue et al. [21]. This network architecture is the same as an unrolled recurrent neural network with some important differences. On a high level, our network is similar to the unfactored architecture in Donahue et al. [21] with each sequence having access to the image features and the previous hidden state in order to predict the next state. We replace the LSTM module with a fully connected layer as in a RNN. However, we also do not use a true recurrent network. The weights for each sequence layer are not shared, and each sequence has access to all the past hidden states. We used 2000 hidden states in our network, but we predict at most six future sequences. We attempted to use recurrent architectures with the publicly available LSTM implementation from Donahue et al. [21]. However, in our experiments they always regressed to a mean trajectory across the data. Our fully connected network has much higher number of parameters than a RNN and therefore highlights the inherent difficulty of this task. Due to the much larger size of the state space, we do not predict optical flow for each and every pixel. Instead, we use kmeans to created a codebook



(a) Input Image (b) Frame 1 (c) Frame 2 (c) Frame 3 (d) Frame 4 (e) Frame 5

Figure 3.8: Qualitative results for multi-frame prediction. The four rows represent predictions from our multi-frame model for future frames. Our extension can predict optical flow over multiple frames.

of 1000 possible optical flow frames, and we predict one of 1000 class as output as each time step. This can be thought of as analogous to a sequential prediction problem similar to caption generation. Instead of a sequence of words, our "words" are clusters of optical flow frames, and our "sentence" is an entire trajectory. We used a set number of sequences, six, in our experiments with each frame representing the average optical flow of one-sixth of a second.

3.5 Conclusion

In this chapter we have presented an approach to generalized event prediction in static scenes. Namely, our framework focuses on motion prediction as a non-semantic form of action prediction. By using an optical flow algorithm to label the data, we can train this model on a large number of unlabeled videos. Furthermore, our framework utilizes the success of deep networks to outperform contemporary approaches to motion prediction.

We find that our network successfully predicts motion based on the context of the scene and the stage of the action taking place. This raises the possibility of incorporating this motion model to predict semantic action labels in images and video. In Chapter 4 we find that the representation learned is useful for human detection. We aim to further explore this route, utilizing representation learning for other recognition tasks. Another possible direction is to utilize the predicted optical flow to predict in raw pixel space, synthesizing a video from a single image. In Chapter 4 we explore this idea with success in some cases.

Chapter 4

Forecasting Pixel Trajectories with Variational Autoencoders

4.1 Introduction

In this chapter, we take the next step and predict pixel motion over longer time frames. Our output space now becomes far more complex. Instead of one frame, we may now have to consider complex trajectories over timescales as long as thirty frames. As such, a classification-as-regression approach is likely not going to be tenable in this context. Regression is a better option here, but how to do account for the multiple possible motion outcomes of an image?

Yuen et al. [128] attempted to approach this problem using nearest neighbors, transferring raw trajectories from multiple matched frames. Unsurprisingly, the algorithm is computationally expensive and fails on testing images which do not have globally similar training images. We propose to revisit the idea of predicting dense trajectories at each and every pixel using a feed-forward convolutional neural network. Using dense trajectories restricts the output space dramatically; this allows our algorithm to learn robust models for visual prediction with the available data. However, the dense trajectories are still high-dimensional, and the output still has multiple modes. In order to tackle these challenges, we propose to use variational autoencoders to learn a low-dimensional latent representation of the output space conditioned on an input image. Specifically, given a single frame as input, our *conditional* variational auto-encoder outputs a mapping from noise variables—sampled from a normal distribution $\mathcal{N}(0,1)$—to output trajectories at every pixel. Thus, we can naively sample values of the latent variables and pass them through the mapping in order to sample different predicted trajectories from the inferred conditional distribution. Unlike other applications of variational autoencoders that generate outputs a priori [36,52,53], we focus on generating them *given the image*. Conditioning on the image is a form of inference, restricting the possible motions based on object location and scene context. Sampling latent variables during test time then allows us to explore the space of possible actions in the given scene.

This chapter makes three contributions. First, we demonstrate that prediction of dense pixel trajectories is a plausible approach to general, non-semantic, self-supervised visual prediction. Second, we propose a conditional variational autoencoder as a solution to this problem, a model that performs inference on an image by conditioning the distribution of possible movements on a scene. Third, we show that our model is capable of learning representations for various recognition tasks with less data than conventional approaches.

4.2 Background

We aim to predict the motion trajectory for each and every pixel in a static, RGB image over the course of one second. Let X be the image, and Y be the full set of trajectories. The raw output space for Y is very large—over four million dimensions assuming a 320x240 image at 30 fps—and it is continuous. We can simplify the output space somewhat by encoding the trajectories in the frequency spectrum in order to reduce dimensionality. However, a more important difficulty than raw data size is that the output space is not unimodal; an image may have multiple reasonable futures.

4.2.1 Model

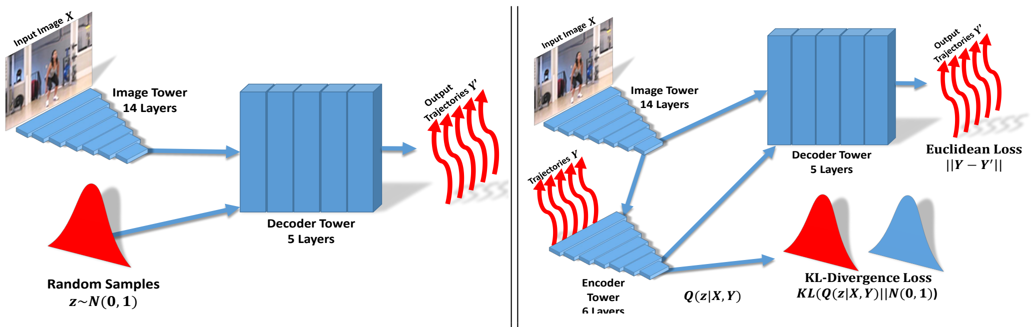
A simple regressor—even a deep network with millions of parameters—will struggle with predicting one-second motion in a single image as there may be many plausible outputs.

Our architecture augments the simple regression model by adding another input z to the regressor (shown in Figure 4.1(a)), which can account for the ambiguity. At test time, z is random Gaussian noise: passing an image as input and sampling from the noise variable allows us to sample from the model's posterior given the image. That is, if there are multiple possible futures given an image, then for each possible future, there will be a different set of z values which map to that future. Furthermore, the likelihood of sampling each possible future will be proportional to the likelihood of sampling a z value that maps to it. Note that we assume that the regressor—in our case, a deep neural network—is capable of encoding dependencies between the output trajectories. In practice, this means that if two pixels need to move together even if the direction of motion is uncertain, then they can simply be influenced by the same dimension of the z vector.

4.2.2 Training by "Autoencoding"

It is straightforward to sample from the posterior at test time, but it is much less straightforward to train a model like this. The problem is that given some ground-truth trajectory Y, we cannot directly measure the probability of the trajectory given an image X under a given model; this prevents us from performing gradient descent on this likelihood. It is in theory possible to estimate this conditional likelihood by sampling a large number of z values and constructing a Parzen window estimate using the resulting trajectories, but this approach by itself is too costly to be useful.

Variational autoencoders [17,36,52,53] modify this approach and make it tractable. The key insight is that the vast majority of samples z contribute almost nothing to the overall likelihood of Y. Hence, we should instead focus only on those values of z that are likely to produce values close to Y. We do this by adding another pathway Q, as shown in Figure 4.1(b), which is trained to map the output Y to the values of z which are likely to produce them. That is, Q is trained to "encode" Y into the latent z space such that the values can be "decoded" back to the trajectories. The entire pipeline can be trained end-to-end using reconstruction error. An immediate objection one might raise is that this is essentially "cheating" at training time: the model sees the values that it is trying to predict, and may just copy



(a) Testing Architecture (b) Training Architecture

Figure 4.1: Overview of the architecture. During training, the inputs to the network include both the image and the ground truth trajectories. A variational autoencoder encodes the joint image and trajectory space, while the decoder produces trajectories depending both on the image information as well as output from the encoder. During test time, the only inputs to the decoder are the image and latent variables sampled from a normal distribution.

them to the output. To prevent the model from simply copying, we force the encoding to be lossy. The Q pathway does not produce a single z, but instead, produces a distribution over z values, which we sample from before decoding the trajectories. We then directly penalize the information content in this distribution, by penalizing the $\mathcal{KL}$-divergence between the distribution produced by Q and the trajectory-agnostic $\mathcal{N}(0,1)$ distribution. The model is thereby encouraged to extract as much information as possible from the input image before relying on encoding the trajectories themselves. Surprisingly, this formulation is a very close approximation to maximizing the posterior likelihood $P(Y|X)$ that we are interested in. In fact, if our encoder pathway Q can estimate the exact distribution of z's that are likely to generate Y, then the approximation is exact.

4.3 Algorithm

4.3.1 The Conditional Variational Autoencoder

We now show mathematically how to perform gradient descent on our conditional VAE. We first formalize the model in Figure 4.1(a) with the following formula:

$$Y = \mu(X, z) + \epsilon \tag{4.1}$$

where $z \sim \mathcal{N}(0,1)$, $\epsilon \sim \mathcal{N}(0,1)$ are both white Gaussian noise. We assume μ is implemented as a neural network.

Given a training example (X_i, Y_i), it is difficult to directly infer $P(Y_i|X_i)$ without sampling a large number of z values. Hence, the variational "autoencoder" framework first samples z from some distribution different from $\mathcal{N}(0,1)$ (specifically, a distribution of z values which are likely to give rise to Y_i given X_i), and uses that sample to approximate $P(Y|X)$ in the following way. Say that z is sampled from an arbitrary distribution $z \sim Q$ with p.d.f. $Q(z)$. By Bayes rule, we have:

$$E_{z\sim Q}\left[\log P(Y_i|z, X_i)\right] = E_{z\sim Q}\left[\log P(z|Y_i, X_i) - \log P(z|X_i) + \log P(Y_i|X_i)\right] \tag{4.2}$$

Rearranging the terms and subtracting $E_{z\sim Q} \log Q(z)$ from both sides:

$$\begin{split}\log P(Y_i|X_i) - E_{z\sim Q}\left[\log Q(z) - \log P(z|X_i, Y_i)\right] = \\ E_{z\sim Q}\left[\log P(Y_i|z, X_i) + \log P(z|X_i) - \log Q(z)\right]\end{split} \tag{4.3}$$

Note that X_i and Y_i are fixed, and Q is an arbitrary distribution. Hence, during training, it makes sense to choose a Q which will make $E_{z\sim Q}[\log Q(z) - \log P(z|X_i, Y_i)]$ (a $\mathcal{KL}$-divergence) small, such that the right hand side is a close approximation to $\log P(Y_i|X_i)$. Specifically, we set $Q = \mathcal{N}(\mu'(X_i, Y_i),\, \sigma'(X_i, Y_i))$ for functions μ' and σ', which are also implemented as neural networks, and which are trained alongside μ. We denote this p.d.f. as $Q(z|X_i, Y_i)$. We can rewrite some of the above expectations as $\mathcal{KL}$-divergences to obtain the standard variational equality:

$$\begin{split}\log P(Y_i|X_i) - \mathcal{KL}\left[Q(z|X_i, Y_i) \| P(z|X_i, Y_i)\right] = \\ E_{z\sim Q}\left[\log P(Y_i|z, X_i)\right] - \mathcal{KL}\left[Q(z|X_i, Y_i) \| P(z|X_i)\right]\end{split} \tag{4.4}$$

We compute the expected gradient with respect to only the right hand side of this equation, so that we can perform gradient ascent and maximize both sides. Note that this means our algorithm is accomplishing two things simultaneously: it is maximizing the likelihood of Y while also training Q to approximate $P(z|X_i, Y_i)$ as well as possible. Assuming a high capacity Q which can accurately model $P(z|X_i, Y_i)$, this second $\mathcal{KL}$-divergence term will tend to 0, meaning that we will be directly optimizing the likelihood of Y. To perform the optimization, first note that our model in Equation 4.1 assumes $P(z|X_i) = \mathcal{N}(0, 1)$, i.e., z is independent of X if Y is unknown. Hence, the $\mathcal{KL}$-divergence may be computed using a closed form expression, which is differentiable with respect to the parameters of μ' and σ'. We can approximate the expected gradient of $\log P(Y_i|z, X_i)$ by sampling values of z from Q. The main difficulty, however, is that the distribution of z depends on the parameters of μ' and σ', which means we must backprop through the apparently non-differentiable sampling step. We use the "reparameterization trick" [53, 93] to make sampling differentiable. Specfically, we set $z_i = \mu'(X_i, Y_i) + \eta \circ \sigma'(X_i, Y_i)$, where $\eta \sim \mathcal{N}(0, 1)$ and $\circ$ denotes an elementwise product. This makes $z_i \sim Q$ while allowing the expression for z_i to be differentiable with respect to μ' and σ'.

4.3.2 Architecture

Our conditional variational autoencoder requires neural networks to compute three separate functions: $\mu(X, z)$ which comprises the "decoder" distribution of trajectories given images ($P(Y|X, z)$), and μ' and σ' which comprise the "encoder" distribution ($Q(z|X, Y)$). However, much of the computation can be shared between these functions: all three depend on the image information, and both μ' and σ' rely on exactly the same information (image and trajectories). Hence, we can share computation between them. The resulting network can be summerized as three "towers" of neural network layers, as shown in Figure 4.1. First, the "image" tower processes each image, and is used to compute all three quantities. Second is the "encoder" tower, which takes input from the "image" tower as well as the raw trajectories, and has two tops, one for μ' and one for σ', which implements the Q distribution. This tower is discarded at test time. Third is the "decoder" tower, which takes input from the "image" tower as well as the samples for z, either produced by the "encoder" tower (training time) or random noise (test time). All towers are fully-convolutional. The remainder of this section details the design of these three towers.

Image Tower: The first, the image data tower, receives only the 320x240 image as input.

The first five layers of the image tower are almost identical to the traditional AlexNet [58] architecture with the exception of extra padding in the first layer (to ensure that the feature maps remain aligned to the pixels).We remove the fully connected layers, since we want the network to generalize across translations of the moving object. We found, however, that 5 convolutional layers is too little capacity, and furthermore limits each unit's receptive field to too small a region of the input. Hence, we add nine additional 256-channel convolutional layers with local receptive fields of 3. To simplify notation, denote $C(k, s)$ as a convolutional layer with kernel size k and receptive field size s. Denote LRN as a Local Response Normalization, and P as a max-pooling layer. Let $\rightarrow C(k, s)_i \rightarrow C(k, s)_{i+1}$ denote a series of stacked convolutional layers with the same kernel size and receptive field size. This results in a network described as: $C(96, 11) \rightarrow LRN \rightarrow P \rightarrow C(256, 5) \rightarrow LRN \rightarrow P \rightarrow C(384, 3) \rightarrow C(384, 3) \rightarrow C(256, 3)_1 \rightarrow C(256, 3)_2 ... \rightarrow C(256, 3)_{10}$.

Encoder Tower: We begin with the frequency-domain trajectories as input, and downsample them spatially such that they can be concatenated with the output of the image tower. The encoder tower takes this tensor as input and processes them with five convolutional layers similar to AlexNet, although the input consists of output from the image tower and trajectory data concatenated into one input data layer. After the fifth layer, two additional convolutional layers compute μ' and σ'. Empirically, we found that predictions are improved if the latent variables are independent of spatial location: that is, we average-pool the outputs of these convolutional layers across all spatial locations. We use eight latent variables to encode the normalized trajectories across the entire image. At training time, we can sample the z input to the decoder tower as $z = \mu' + \eta \circ \sigma'$ where $\eta \sim \mathcal{N}(0, 1)$. μ' and σ' also feed into a loss layer which computes the $\mathcal{KL}$ divergence to the $\mathcal{N}(0, 1)$ distribution. This results in a network described as: $C(96, 11) \rightarrow LRN \rightarrow P \rightarrow C(256, 5) \rightarrow LRN \rightarrow P \rightarrow C(384, 3) \rightarrow C(384, 3) \rightarrow C(256, 3) \rightarrow C(8, 1) \times 2$.

Decoder Tower: We replicate the sampled z values across spatial dimensions and multiply them with the output of the image tower with an offset. This serves as input to four additional 256-channel convolutional layers which constitute the decoder. The fifth convolutional layer is the predicted trajectory space over the image. This can be summarized by: $C(256,3)_1 \rightarrow C(256,3)_2 ... \rightarrow C(256,3)_4 \rightarrow C(10,3)$. This output is over a coarse resolution, i.e., 16x20 pixels. The simplest loss layer for this is the pure Euclidean loss, which corresponds to log probability according to our model (Equation 4.1). However, we empirically find much faster convergence if we split this loss into two components: one is the *normalized* version of the trajectory, and the other is the magnitude (with a separate magnitude for horizontal and vertical motions). Because the amount of absolute motion varies considerably between images—and in particular, some action categories have much less motion overall—the normalization is required so that the learning algorithm gives equal weight to each image. The total loss function is therefore:

$$\begin{aligned} L(X,Y) = ||Y_{\text{norm}} - \hat{Y}_{\text{norm}}||^2 + ||M_x - \hat{M}_x||^2 + ||M_y - \hat{M}_y||^2 \\ + \mathcal{KL}\left[Q(z|X,Y) \| \mathcal{N}(0,1)\right] \end{aligned} \tag{4.5}$$

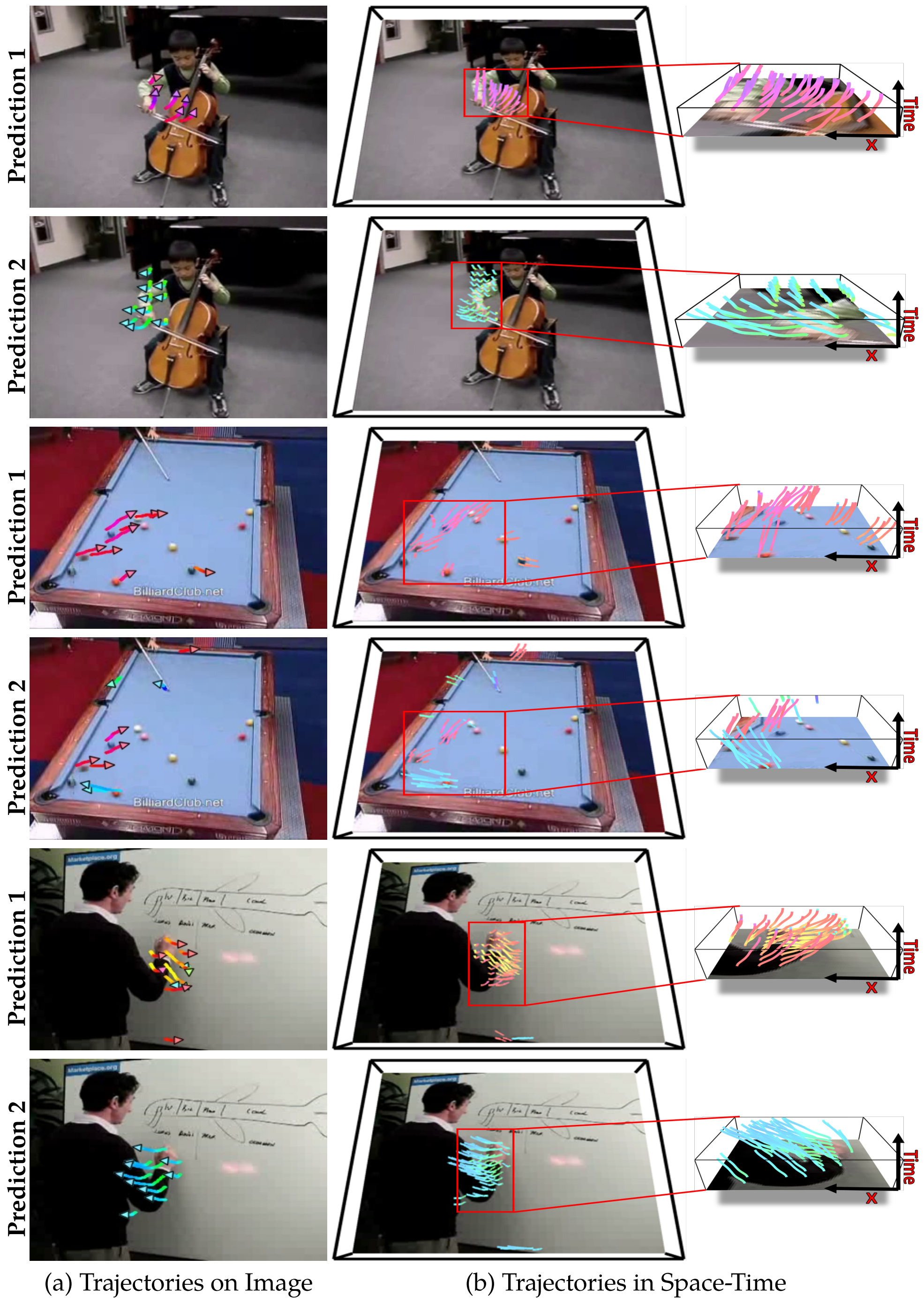
Where Y represents trajectories, X is the image, M_i are the global magnitudes, and $\hat{Y}$, $\hat{M}_i$ are the corresponding estimates by our network. The last term is the KL-divergence loss of the autoencoder. We find empirically that it also helps convergence to separate both the latent variables and the decoder pathways that generate $\hat{Y}_{\text{norm}}$ from the ones that generate $\hat{M}$, perhaps due to the differences in scaling between these two outputs.

Coarse-to-Fine: The network as described above predicts trajectories at a stride of 16, i.e., at $1/16$ the resolution of the input image. This is often too coarse for visualization, but training directly on higher-resolution outputs is slow and computationally wasteful. Hence, we only begin training on higher-resolution trajectories after the network is close to convergence on lower resolution outputs. We ultimately predict three spatial resolutions—1/16, 1/8, and 1/4 resolution—in a cascade manner similar to [24]. The decoder outputs directly to a 16x20 resolution. For additional resolution, we upsample the underlying feature map and concatenate it with the conv4 layer of the image tower. We pass this through 2 additional convolution layers, $D = C(256,5) \rightarrow C(10,5)$, to predict at a resolution of 32x40. Finally, we upsample this feature layer D, concatenate it with the conv1 layer of the image tower, and send it through one last layer of $C(10,5)$ for a final output of 64x80.

Implementation Details: Given videos, we extract one-second (31-frame) clips, use the implementation of Wang et al. [115] to stabilize them respective to the first frame, and generate the trajectories which we use as a label. As the implementation of Wang et al. [115] tracks pixels over different scales, we take the average trajectory over all scales for a given pixel. For each pixel in the first frame of a clip, we encode its trajectory via an x- and y-offset relative to the pixel's start location for each subsequent frame—a 60-dimensional vector. We perform a discrete cosine transform separately for the x and y offsets and take only the first 5 components of each. We use batch normalization [44] to train the network, adding a batch normalization layer after every convolution layer that does not produce an output where scale is meaningful (i.e. μ, μ', σ').

4.4 Experiments

Because almost no prior work has focused on motion prediction beyond the timescale of optical flow, there are no established metrics or datasets for the task. For our quantitative evaluations, we chose to train our network on videos from the UCF-101 dataset [101]. Although there has been much recent progress on this dataset from an action recognition standpoint, pixel-level prediction on even the UCF-101 dataset has proved to be non-trivial [90, 102]. Because the scene diversity is low in this dataset, we utilized as much training data as possible, i.e., all the videos except for a small hold out set for every action. We sampled every 3rd frame for each video, creating a training dataset of approximately 650,000 images. Testing data for quantitative evaluation came from the testing portion of the THUMOS 2015 challenge dataset [35]. The UCF-101 dataset is the training dataset for the THUMOS challenge, and thus THUMOS is a relevant choice for the testing set. We randomly sampled 2800 frames and their corresponding trajectories for our testing data. We will make this list of frames publicly available. We use two baselines for trajectory prediction. The first is a direct regressor for trajectories using the same layer architecture from the image data tower. The second baseline is the optical flow prediction network from Walker et al. [114] which was trained on the same dataset. We extrapolate the network predictions over one second.



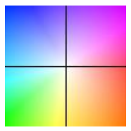
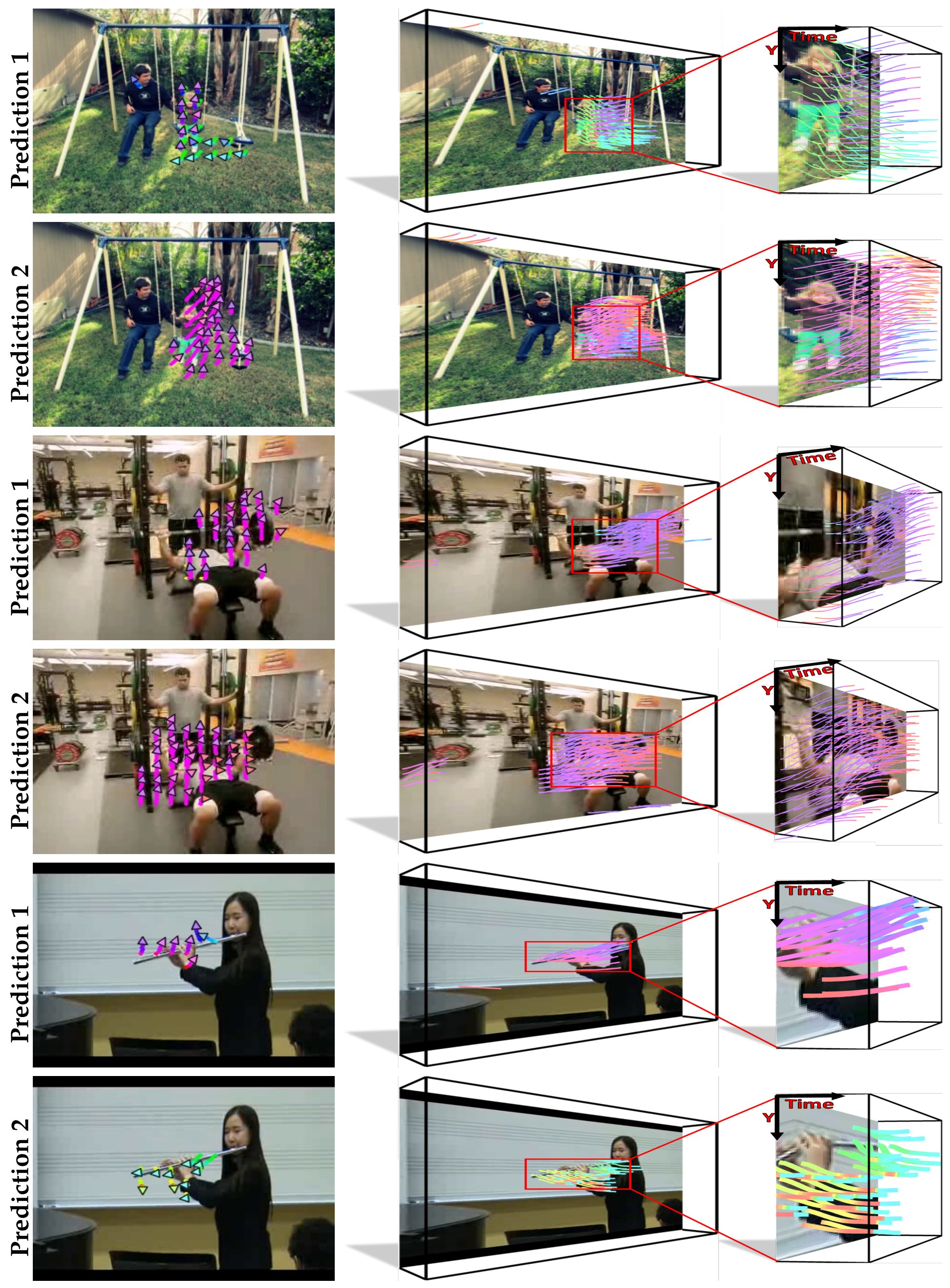


Figure 4.2: Predictions of our model based on clustered samples. The directions of the trajectories at each point in time are color-coded according to the square on the right. On the right is a full view of two predicted motions in 3D space-time; on the left is the projection of the trajectories onto the image plane. Best seen in video.



(a) Trajectories on Image (b) Trajectories in Space-Time

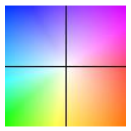


Figure 4.3: Predictions of our model based on clustered samples. The directions of the trajectories at each point in time are color-coded according to the square on the right. On the right is a full view of two predicted motions in 3D space-time; on the left is the projection of the trajectories onto the image plane. Best seen in video.

Table 4.1: Quantitative Results on the THUMOS 2015 Dataset. Lower is better.

Method	Negative Log Likelihood
Regressor	11463
Optical Flow [114]	11734
Ours	**11082**

4.4.1 Quantitative Results - Log Likelihood

Choosing an effective metric for future trajectory prediction is challenging since the problem is inherently multi-modal. There might be multiple correct trajectories for every testing instance. Simple metrics like Euclidean distance from the ground truth become difficult to interpret in this situation: the optimal prediction for such a metric would be one which lies in-between the possibilities, a prediction which is not necessarily sensible in itself. We thus first evaluate our method in the context of generative models: we evaluate whether our method estimates a distribution where the ground truth is highly probable. Namely, given a testing example, we estimate the full conditional distribution over trajectories and calculate the log-likelihood of the ground truth trajectory under our model. For log-likelihood estimation, we construct Parzen window estimates using samples from our network, using a Gaussian kernel. In our evaluations, we compared trajectories on the coarse resolution output—$10 \times 16 \times 20$—resulting in a 3200-dimensional vector space. We estimate the optimal bandwidth for the Parzen window via gridsearch on the training data. As the networks were originally trained to optimize over normalized trajectories and magnitude separately, we also separate normalized trajectory from magnitude in the testing data, and we estimate bandwidths separately for normalized trajectories and magnitudes. We used 800 samples per image for Parzen window estimation.

To evaluate the log-likelihood of the ground truth under our first baseline—the regressor—we treat the regressor's output as a mean of a multivariate Gaussian distribution. In order to obtain an upper bound on the log-likelihood for the regressor, we optimize the bandwidths (i.e. the standard deviation parameters for the direction and magnitude, which are shared across the entire dataset) over the testing data. We then estimate the log-likehood of the ground-truth trajectory under this distribution. The optical flow network uses a soft-max

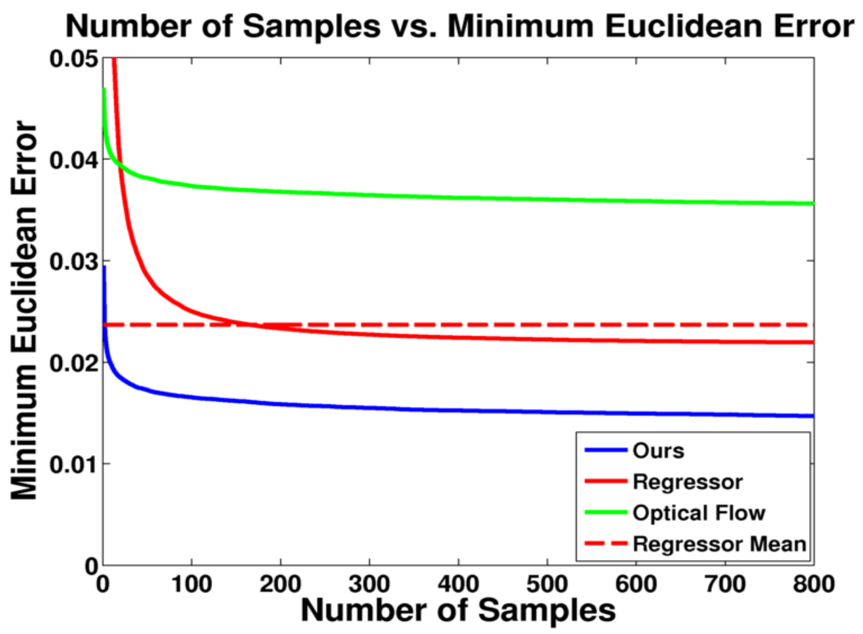


Figure 4.4: Average Minimum Euclidean distance for each method. We take the closest prediction in each testing image and plot the average of these distances as the number of samples grows per image.

layer to estimate the per-pixel distribution of motions in the image; we thus take samples of motions using this estimated distribution. We then use the samples to estimate a density function in the same manner as the VAE. In the same way for the regressor, we optimize the bandwidth over the testing data in order to obtain an upper bound for the likelihood.

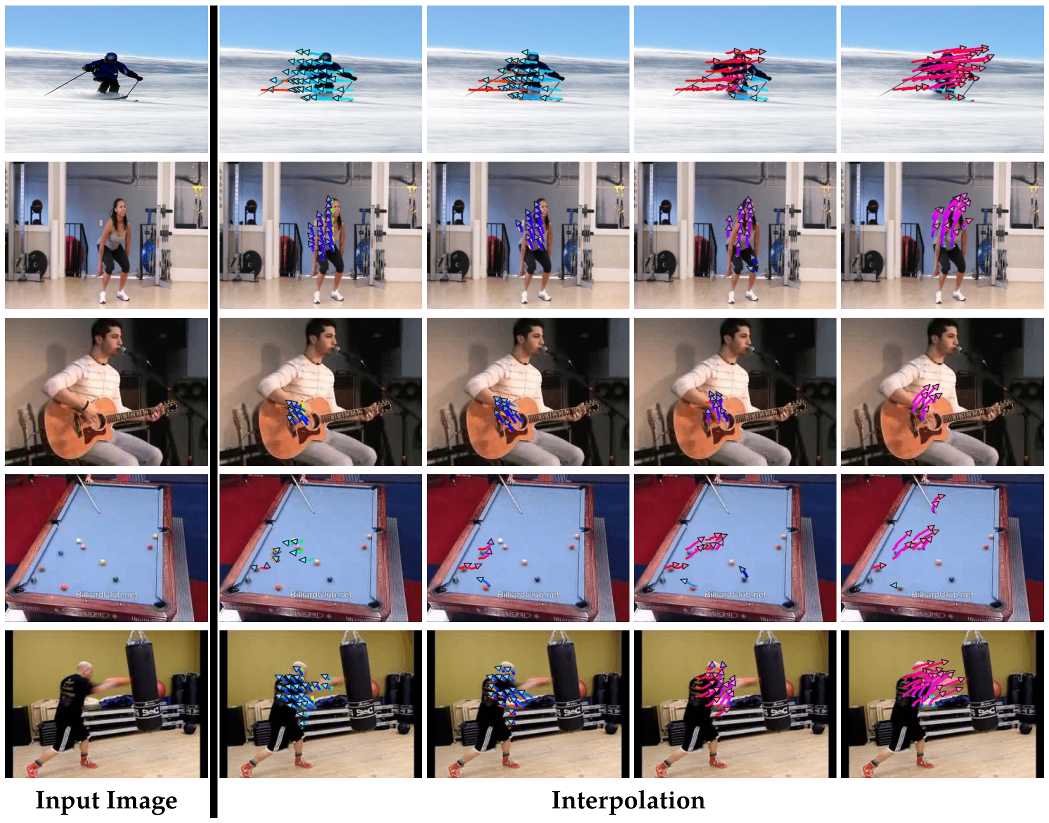
In Table 4.1, we show our evaluations on the baselines for trajectory prediction. Based on the mean log-likelihood of the ground-truth trajectories under each model, our method outperforms a regressor trained on this task with the same architecture as well as extrapolation from an optical flow predictor. This is reasonable since the regressor is inherently unimodal: it is unable to predict distributions where there may be many reasonable futures, which Figures 4.2 and 4.3 suggest is rather common. Interestingly, extrapolating the predicted optical flow from [114] does not seem to be effective, as motion may change direction considerably even over the course of one second.

4.4.2 Quantitative Results - Euclidean Distance

As log-likelihood may be difficult to interpret, we use an additional metric for evaluation. While average Euclidean distance over all the samples in a particular image may not be particularly informative, it may be useful to know what was the best sample created by the algorithm. Specifically, given a set number n of samples per image, we measure the Euclidean distance of the closest sample to the ground truth and average over all the testing images. For a reasonable comparison, it is necessary to make sure that every algorithm has an equal number of chances, so we take precisely n samples from each algorithm per image. Our framework can naturally output multiple predictions. For the optical flow baseline [114], we can take samples from the underlying softmax probability distribution. For the regressor, we sample from a multivariate Gaussian centered at the regressor output and use the bandwidth parameters estimated from grid-search. We plot the average minimum Euclidean distance for each method in Figure 4.4. We find that even with a small number of samples, our algorithm outperforms the baselines. The additional dashed line is the result from simply using the regressor's direct output as a mean, which is equivalent to sampling with a variance of 0. Note that given a single sample, the regressor outperforms our method since it directly optimized the Euclidean distance at training time. Given more than a few samples, however, ours performs better due to the multimodality of the problem.

4.4.3 Qualitative Results

We show some qualitative results in Figures 4.2 and 4.3. For these results, we cluster 800 samples into 10 clusters via Kmeans and show top two clusters with significant motion. Our method is able to identify active objects in the scene whether they are hands, entire bodies or objects such as billiard balls. The network then predicts motion based on the the context of the scene. For instance, the network tends to predict up and down motions for the man lifting a weight and the people on the swing in Figure 4.3. The boy playing the violin moves his arm left and right, and the man writing on the board moves his arm across the board. Figure 4.5 shows the role latent variables play in predicting motion in some selected scenes



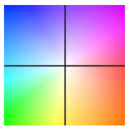


Figure 4.5: Interpolation in latent variable space between two points from left to right. Each column represents a set of images with the same latent variables. Left to right represents a linear interpolation between two points in z-space. The latent variables influence direction to some extent, but the context of the image either amplifies or greatly reduces this direction. The squatting woman is always moving upward, but the skier changes drastically in direction.

with a distinct action: interpolating between latent variable values interpolates between the motion prediction. Based on this figure, at least some latent variables encode the direction of motion. However, the network still depends on image information to restrict the types of motions that can occur in a scene. Given a set of the exact same latent variables, the predicted motion is modulated based on the action in the scene. For instance, the man skiing moves only left or right, while the woman squatting largely moves only up or down with small changes to the x-axis.

Table 4.2: mean Average Precision (mAP) on VOC 2012. The "External data" column represents the amount of data exposed outside of the VOC 2012 training set. "cal" denotes the between-layer scale adjustment [57] calibration.

VOC 2012 test	external data	aero	bike	bird	boat	bottle	bus	car	cat	chair	cow	table	dog	horse	mbike	person	plant	sheep	sofa	train	tv	mAP
scratch+cal	N/A	67.5	49.8	27.9	23.9	13.6	57.8	48.1	51.7	16.1	33.2	29.2	45.3	51.9	58.8	51.7	16.8	39.7	29.4	55.7	43.5	40.6
kmeans [57]+cal	N/A	71.1	56.8	31.8	28.1	17.7	62.5	56.6	59.9	19.9	37.3	36.2	52.9	56.4	64.3	57.1	21.2	45.8	39.1	60.9	46.0	46.1
rel. pos. [19]+cal	1.2M ImageNet	**74.3**	**64.7**	**42.6**	**32.6**	25.9	**66.5**	**60.2**	**67.9**	**27.0**	**47.9**	**41.3**	**64.5**	**63.4**	**69.1**	57.5	25.3	51.9	**46.7**	**64.6**	51.4	**52.3**
egomotion [1]+cal	20.5K KITTI img.	70.7	56.3	31.9	25.6	18.7	60.4	54.1	57.6	19.8	40.9	31.8	51.9	54.9	61.7	53.5	19.8	45.2	36.3	56.9	49.1	44.9
vid. embed [118]	1.5M vid. frames (100k vid)	68.8	62.1	34.7	25.3	**26.6**	57.7	59.6	56.3	22.0	42.6	33.8	52.3	50.3	65.6	53.9	**25.8**	51.5	32.3	51.7	51.8	46.2
vid. embed [118]	5M vid. frames (100k vid)	69.0	64.0	37.1	23.6	24.6	58.7	58.9	59.6	22.3	46.0	35.1	53.3	53.7	66.9	54.1	25.4	**52.9**	31.2	51.9	51.8	47.0
vid. embed [118]	8M vid. frames (100k vid)	67.6	63.4	37.3	27.6	24.0	58.7	59.9	59.5	23.7	46.3	37.6	54.8	54.7	66.4	54.8	**25.8**	52.5	31.2	52.6	**52.6**	47.5
vid. embed [118]+cal	8M vid. frames (100k vid)	68.1	53.1	31.9	24.3	16.9	57.2	50.8	58.4	14.1	36.9	27.6	52.5	49.6	60.0	48.4	15.8	41.9	34.4	55.6	45.6	42.2
ours+cal	13k UCF-101 vid.	71.7	60.4	34.0	27.8	18.6	63.5	56.6	61.1	21.2	39.3	35.1	57.1	58.6	66.0	**58.4**	20.5	45.6	38.3	62.1	49.9	47.3

4.4.4 Representation Learning

Prediction implicitly depends on a number of fundamental vision tasks: for example, the network must recognize the scene to infer the action that is being performed and must detect, localize, and infer the pose of humans and objects that may move. Hence, we expect the representation learned for the task of motion prediction may generalize for other vision tasks. We thus evaluate the representation learned by our network on the task of object detection. We take layers from the image tower and fine-tune them on the PASCAL 2012 training dataset. For all methods, we apply the between-layer scale adjustment [57] to calibrate the pre-trained networks, as it improves the finetuning behavior of all methods except one. We then compare detection scores against other unsupervised methods of representation learning using Fast-RCNN [33]. We find that from a relatively small amount of data, our method outperforms other methods that were trained on datasets with far larger diversity in scenes and types of objects. Interestingly, our method outperforms all unsupervised methods, even Doersch et al. [19], on human detection, likely because most of the movement our algorithm needs to predict comes from humans.

4.5 Conclusion

In this chapter we have presented an approach to predict dense trajectories from pixels. Specifically, our framework proposes a variational autoencoder conditioned on images: the framework uses latent variables (and the predicted distribution) to represent multiple possible trajectories. Our method requires no human labels and can be trained efficiently with back-propagation. Furthermore, we showed that our method learns a representation that transfers to other vision tasks such as object detection. As motion information is essential to action recognition [99,115], we aim in the future to explore the use of motion prediction in action recognition tasks.

Chapter 5

Unsupervised Prediction with Mid-Level Elements

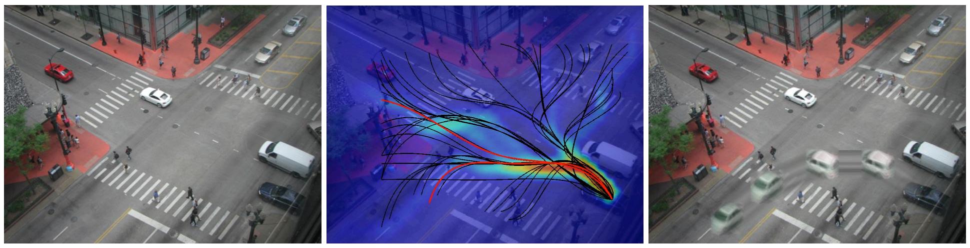

(a) Input Image (b) Trajectory Heatmap (c) Predicted Event

Figure 5.1: Consider the scene shown in image (a). In this chapter, we discover active elements in the scene and detect them. Consider the blue car in the lower left corner. Using a trained motion model, we can then create a heatmap of the possible trajectories this element can take (b). Finally, we can visualize the appearance change of this element over time (c).

5.1 Introduction

Building upon the recent success of mid-level elements [100], in this chapter, we propose a framework for visual prediction which uses these mid-level elements as building blocks of prediction. In our framework, we model not only the movement and transitions of these elements in the scene but also how the appearances of these elements can change. Our new framework has the following advantages over previous approaches. First, our approach makes no assumption about what can act as an agent in the scene. It uses a data-driven approach to identify the possible agents and their activities. Second, using a patch-based representation allows us to learn the models of visual prediction in a completely unsupervised manner. We also demonstrate how a rich representation allows us to use a simple non-parametric approach to learn a state-of-the-art visual prediction model. Finally, because our approach exploits mid-level elements instead of full scenes for creating associations, it allows for generalization and sharing across different instances.

5.2 Our Approach

Given an input scene, our goal is to predict what is going to happen next—what parts of the image are going to remain the same, what parts of the image are likely to move, and how they move. The central idea is that scenes are represented as a collection of mid-level elements, detected using a sliding window, where agents can either move in space or change visual appearances. Each agent is predicted independently assuming a static scene. We model the distribution over the space of possible actions using a transition matrix which represents how mid-level elements can move and transition into one another and with what probability. For example, an element that represents a frontal car can transition to a patch facing right if the car turns. Given the mid-level elements and their possible actions, we first determine which is the most likely agent and the most likely action given the scene. However, this notion of most likely action depends upon goals and the context/scene around the elements. For example, in Figure 5.1, the visual prediction of a car not only depends upon the goal but also on the other cars, pedestrians, and the sidewalk in the image. Therefore, as a next step, we need to model the interaction between the active element (agent) and its surrounding. We model this interaction using a reward function $\psi_i(x, y)$ which models how likely is it that an element of type i can move to location (x, y) in the image. For example, a car element will have high reward for road-like areas and low reward for grass-like areas—without modeling semantics explicitly. Given a goal, our approach then infers the most likely path using the transition matrix and computed reward (Section 5.2.4). Finally, if the goal is unknown—which is the case here, we propose to sample several goals and select

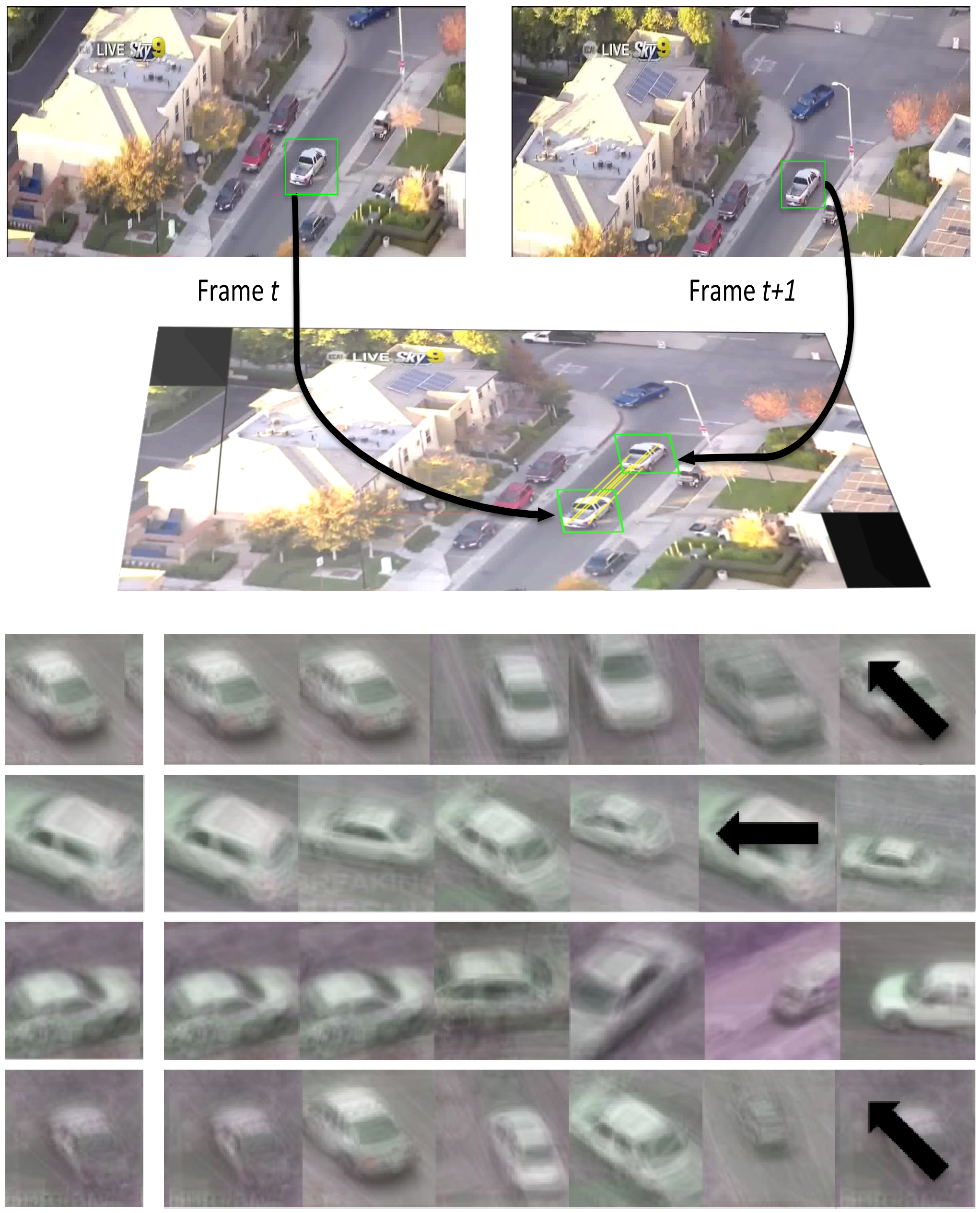


Figure 5.2: On the top is an illustration of patch mapping. Two frames are matched using an estimated homography, and KLT features inside the bounding boxes of detections in each frame direct patch movements. On the bottom are top possible transitions learned from training data. The left-most column are the original elements, and other columns are possible transitions. Note each element can either change appearance and morph into another element, or it can just move in space (arrowed squares). The elements are shown as average images of top detections on the training data.

the most likely goal based on high expected reward.

During training, we need to learn the mid-level representation, the space and likelihood of transitions for each element, and the reward function $\psi_i(x, y)$ for every possible element. We propose to learn these from large quantities of spatio-temporal visual data in an unsupervised manner. We first create a state space of mid-level patches that distinguish the domain from the rest of the visual world. Within a domain such as videos of cars driving on roads or pedestrians walking outdoors, we first apply the work of Singh et al. [100] to extract mid-level elements. These elements are visually meaningful and discriminative HOG clusters trained against a large set of general visual data. Each element can act as an agent which can move. Instead of using domain based assumptions such as agents being cars or humans, our approach exploits data to decide which features are significant and extract the agents. For example, in the case of the VIRAT dataset [80], one of the elements groups two people since the two people are likely to move together and hence can be modeled as a single agent. Once we have extracted the dictionary of mid-level elements for a given domain, we use temporal information to find patch-to-patch transitions as well as their spatial behavior on the image plane (Section 5.2.1). We can use the statistics of the transition matrix to determine which elements are the active agents in the scene. Finally, we learn a reward function over the state space which is combined with transition matrix to infer the predictions (Section 5.2.2).

5.2.1 Learning The Transitions

Given the dictionary (Figure 5.2) of mid-level elements, the first step is to learn a temporal model over these elements. The temporal model is represented using a transition matrix where element i can either move in one of the eight directions (top, left, bottom, right, top-left, top-right, bottom-left, bottom-right) or transition into another element in the dictionary. How do we learn these transitions? Given the training data, we extract pairs of frames at least a second apart and detect mid-level patches in both the frames. To learn the transition we need to obtain the correspondence between the detections in the two frames. We obtain this correspondence by counting the number of common features tracked using

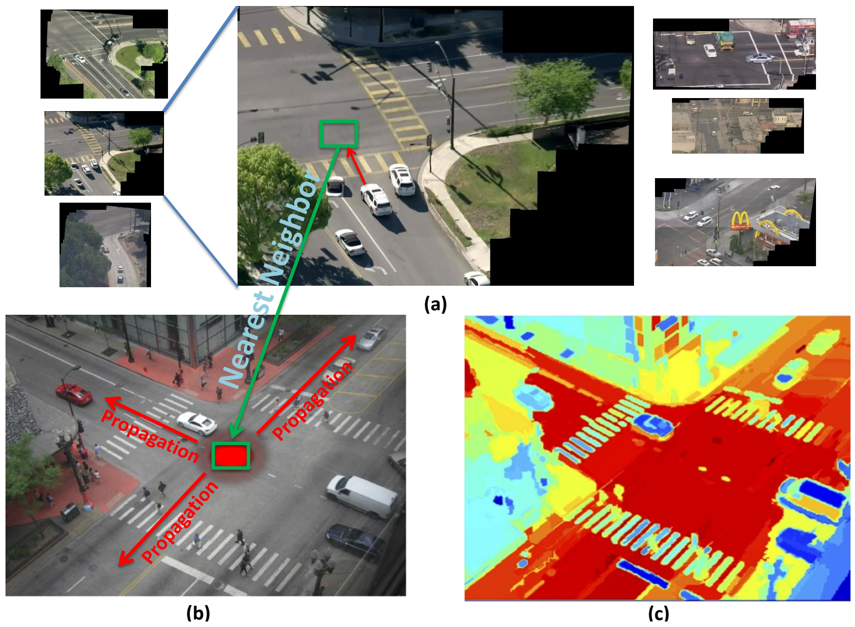


Figure 5.3: A reward function (c) is propagated by taking texture information from the destinations of observed moving patches in training data (a). During test time (b), the area of the image with the closest texture to the training textures is set as the highest reward in the scene. Other areas of the scene (via graphcut segments) are scored according to the similarity to the chosen window. Warm colors indicate high reward; cooler colors indicate low reward.

the KLT Tracker [73] inside the two bounding boxes.

We interpret the mapping as either an appearance or spatial transition. If the patches are of two cluster types, then the mapping is counted as a transition from one cluster type to another regardless of spatial movement. For a patch to be counted as a spatial movement on the image plane, the mapped patches must be of the same type, and they must not overlap. In order to compensate for camera motion these movements are computed on a stitched panorama obtained via SIFT matching [70]. For each transition, we normalize for total number of observed patches as well. This gives us the probability of transition for each mid-level patch. Figure 5.2 shows some of the top transitions for four mid-level elements.

5.2.2 Learning Contextual Information

A transition matrix captures the most likely action in absence of the contextual information. For example, a car facing right is most likely to move right. However, the actions of agents are not only dependent on the likely transitions but also on the scene and the surroundings in which they appear. For example, if there is a wall in front of the car, it is unlikely to move in that direction. Therefore, apart from capturing the statistics of patch transitions, we need information about how a patch may interact with its environment. We model these interactions using a reward function $\psi_i(x, y)$ which models how likely is it that an element of type i can move to location (x, y) in the image. Because each element is supposed to represent a different underlying concept, we learn a separate reward function for the interaction of each element within the scene.

We use a non-parametric approach over segments to model the reward function. To obtain the training data for reward function of element type i, we detect the element in the training videos and observe which segments are likely to overlap with that element in time. For example, the car elements are likely to overlap with road segments, and hence those road segments act as instances of positive reward areas for car element. Using such instances, we build a training set for every element in the dictionary. Once we have the training sets for every patch type i, we can use this to compute the reward function at test time. Each segment in the test image retrieves the nearest neighbor using Euclidean distance over image features. We choose the top-N nearest neighbors to label high reward areas in the image and then propagate the reward function within the image based on visual similarity — graphcut segments which look similar to high reward regions in the query image also get high reward. Figure 5.3 shows an example of reward propagation.

5.2.3 Inferring Active Entities

Once we have learned the transition function and reward function $\psi_i(x, y)$ for every mid-level element, we can predict what is going to happen next. The first step of prediction inference requires estimating the elements in the scene that are likely to be active. Kitani et al. [54] choose the active agents manually. In this work, we propose an automatic approach to infer the likely active agent based on the learned transition matrix. Our basic idea is to rank cluster types by their likelihood to be spatially active. We assume the active agents are the elements that are likely to move themselves, likely to transition to patches that can move, and in a scene that allows the element to move to high reward areas in its neighborhood. In order to detect the top possible elements in a scene that satisfy these properties, we first detect the instances of each element using sliding-window detection. We then rank these instances based on contextual information. The context-score for a patch i at location (x, y) is given by

$$\sum_d p_i^d e^{\psi_i(x+d_x, y+d_y)} \tag{5.1}$$

where $d = (d_x, d_y)$ is the direction of the movement, p_i^d is the transition probability in direction d and $\psi_i(x + d_x, y + d_y)$ computes the reward for moving the patch from (x, y) to $(x + d_x, y + d_y)$. In this chapter, we discretize d into eight directions.

Instead of predicting all the elements discovered, we only predict the activities of elements that are likely to change location either directly or by transition. We compute the likelihood of changing location based on the transition matrix. Therefore, the elements which have high movement transition likelihood or transition to a element that has high movement likelihood are selected.

5.2.4 Planning Actions and Choosing Goals

Once we have selected the most likely active agents, we use the transition matrix combined with the reward function to search for optimal actions/transitions given a spatial goal in the scene. We first re-parameterize the reward function $\psi_i(x, y)$ such that if the state is $s = (x, y, i)$ (patch i being at location (x, y)), then the reward is $\phi(s)$. Each decision a is quantified by the expected reward: *i.e.*, the product of the probability of the decision p_a and the reward function $\phi(s)$ in the new state. Note that the decision can either be a movement or be a transition. In the first case, the location (x, y) changes in the state while in the second case, we have the same location by a new cluster type.

Our goal is to find the optimal set of actions/decisions $\sigma = (a_1, ..a_n)$, such that these actions maximize expected reward (minimize cost), and these actions reach the goal state

g. We formulate this as maximization of the reward function

$$\max_{\sigma} \sum_{a_t \in \sigma} p_{a_t} \phi(s_{t+1}) \quad \text{s.t.} \;\; \sigma \odot s_0 = g \tag{5.2}$$

where s_0 is the initial state and $\odot$ operator applies a set of actions to a state to estimate goal state. We then use Dijsktra's algorithm to plan a sequence of optimal decisions σ from an initial state to all given goal states by *converting rewards to costs*. Specifically, we create a graph where each state is represented as a node in the graph. For example, for a 100x100 image and dictionary size of 750 elements, there will be 100x100x750 nodes in the graph. The edges between the nodes represent the cost of transitioning from state s_i to s_j. This cost depends on the transition probabilities and rewards. Given this graph, the initial state is represented as the source node in the graph, and the goal nodes are considered to be along the edge of image. We then run Dijsktra's algorithm to get the optimal path. We select the best path among different goals based on average expected reward — normalized with respect to the total number of decisions.

5.2.5 Implementation Details

KLT Tracker: We use the Kanade-Lucas tracking algorithm on extracted SURF features [5] to track how detected patches move in each scene. Given detected patches in two frames, we track the SURF features which initially lie inside the bounding box of a given patch.

Reward Function: The distance metric for the reward function is computed using 69-dimensional feature vector based on RGB and a bag of words.

Other Details: The selected frames during transition matrix learning were 4 seconds apart in the VIRAT dataset and only one second apart in the car chase dataset due to faster motion.

Experiment	NN + Sift-Flow	[54]	Ours
No Agent Given			
Mean Distance	22.34	-	**14.38**
Median Distance	16.68	-	**10.91**
Agent Given			
Mean	27.55	37.94	**21.55**
Median	23.77	30.23	**14.98**

Figure 5.4: Mean and Median of the error of closest path over 44 videos in the car chase dataset for no agent given, and Mean and Median error of the top-ranked path for a given agent.

5.3 Experimental Results

Because there has been little work in the field of visual prediction, there are no established datasets, baselines, or evaluation methodologies. We perform extensive qualitative and quantitative evaluation for path prediction, and we provide detailed qualitative analysis for prediction of visual appearances.

Baselines: There are no algorithms for unsupervised visual prediction; therefore we compare against nearest neighbor followed by sift-flow warping [67, 128] and the max-entropy based Inverse Optimal Control (IOC) based algorithm of Kitani et al. [54]. For the NN baseline, we use a Gist-matching [81] approach similar to that of Yuen et. al. [128]. We then use the labeled path from the nearest-neighbor as the predicted trajectory and warp it into the scene using Sift Flow [67]. For Kitani et al. [54], we first learn a reward function using IOC, and then given the initial agent we predict the most likely paths using a Markov Decision Process (MDP).

Datasets: We perform experiments on two different datasets: a Car Chase Dataset (collected from YouTube) and the VIRAT dataset [80].

Evaluation Metric: We use the modified Hausdorff distance (MHD) from [54] as a measure of the distance between two trajectories. The MHD allows for local time warping by finding the best local point correspondence over a small temporal window (3 steps in our experiments). Each algorithm will generate a collection of likely predicted paths and therefore we

VIRAT	Ours	MDP (Our Reward)	[54]
Mean	**108.81**	128.48	147.32
Median	**77.79**	99.05	150.24

Figure 5.5: Mean and median of the error of top predicted path.

will compute the distance between the top-N generated predictions and the ground-truth path.

5.3.1 Car Chase Dataset

For our main experiments, we created a new dataset by downloading videos from Youtube of aerial car chase videos. In total we used 183 videos (from 48 different scenes) that lasted from 5 to 30 seconds. We used 139 videos from 37 scenes for training and 44 videos from 11 scenes for testing. For extracting discriminative mid-level elements, we used 1871 random frames from the training set in addition to 309 outdoor scenes from Flickr as the discovery dataset, and we use the MIT Indoor 67 [88] dataset for the negative dataset. We manually annotated the trajectories of the car in 44 test videos which were used as the ground-truth to evaluate the algorithm.

Qualitative: Figure 5.7 shows some qualitative results. Notice how the reward function captures that the road is a high reward region for the car patch. The bus in the top image and the cars in the bottom image are the areas of low reward. Similarly, sidewalk is also considered a low reward area. Given this reward function, our inference algorithm generated the possible paths, and the marginalization of these paths and some sampled paths are shown in the figure. Finally, notice the visual predictions in (d) and (h). Notice how the car turns in the top image avoiding the bus and how the car maneuvers between the two cars in the bottom image. Figure 5.8 shows some more qualitative examples of visual prediction generated by our algorithm.

Quantitative: In the first experiment, we compare the performance of our approach with the NN-baseline when no agents are given. In this way, we are measuring the ability of our method not only to find the correct active entity in the scene but also effectively predict its

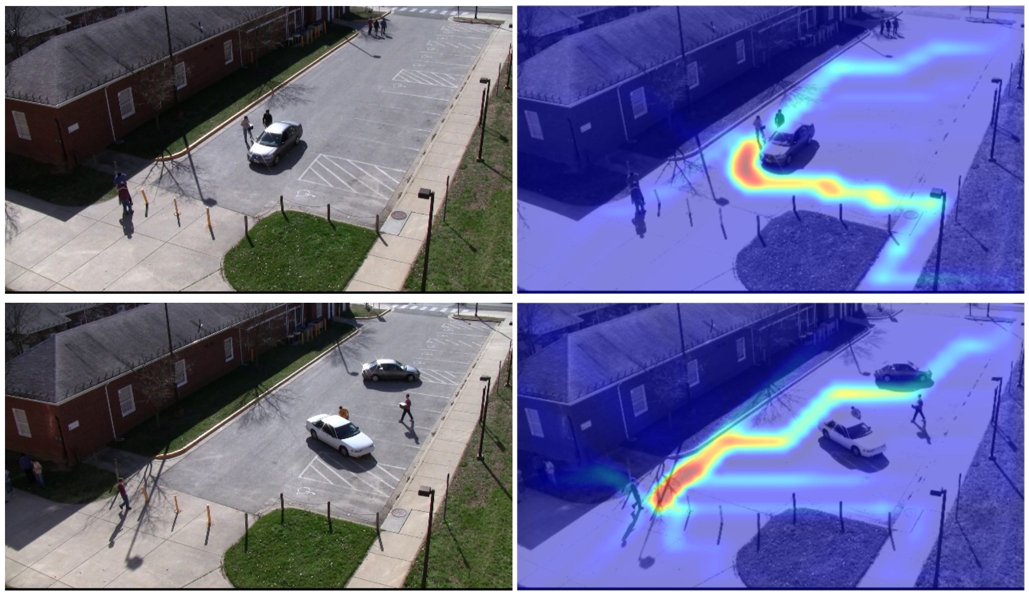
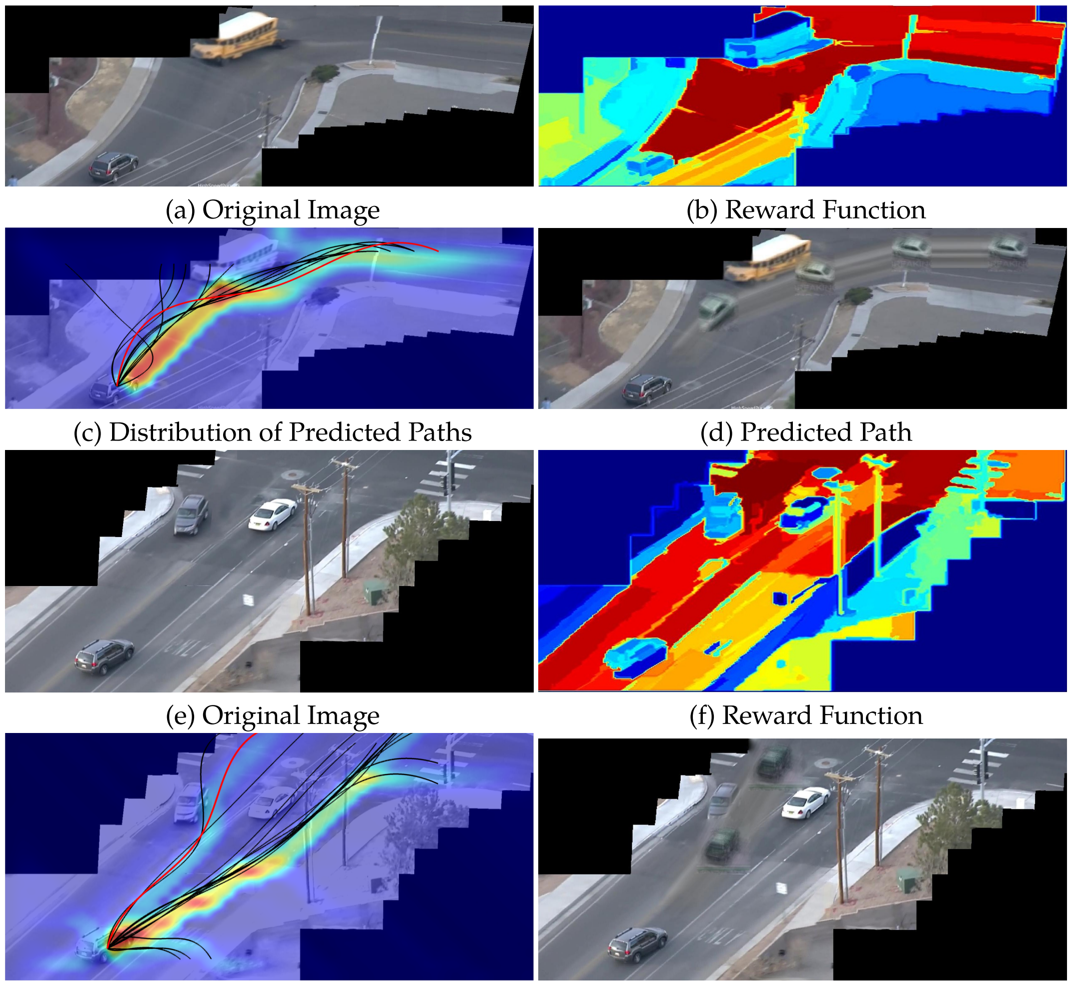

(a) Original Image (b) Prediction Heatmap

Figure 5.6: Qualitative predictions for our approach on the VIRAT dataset. The left shows the original image; the right shows a heatmap of possible paths.

spatial activity. We use our algorithm to identify top three active agents in the scene and predict two paths per agent. We also allow the NN algorithm to generate six paths using top-6 matches. In order to compare path distance across test instances, all query panoramas are resized to a canonical size where the smallest dimension is 50 pixels. Table 5.4(top) shows the performance of the approach. Considering the median error, we show 35% improvement over the baseline. The ground truth car was in the method's top three active entities in 73% of cases.

In the second experiment, we ask a different question: given a manually selected agent, how well can we estimate the overall distribution of possible actions? In this case we can add Kitani et al. [54] as a second baseline. The paths are ranked according to the expected reward. Since the agent is given, we compare the top paths in this case, not the entire distribution of paths as in Kitani et al. [54]. Table 5.4(bottom) shows the that our approach again is superior to the NN-baseline and even shows improvement over Kitani et al. [54]. IOC in this case appears to do poorly because of the underlying semantic features.


(a) Original Image (b) Reward Function

(c) Distribution of Predicted Paths (d) Predicted Path

(e) Original Image (f) Reward Function

(g) Distribution of Predicted Paths (h) Predicted Path

Figure 5.7: Some qualitative predictions of our approach. The upper right images (b,f) represents a reward function for a given car patch over the image. The lower left (c,g) shows a heatmap of possible locations where the agent can be and some of predicted trajectories, and the lower right (d,h) demonstrates the visualization of one such trajectory.

5.3.2 VIRAT Dataset

For our second dataset, we chose a subset of the VIRAT dataset corresponding to a single scene A used in Kitani et al. [54]. Since VIRAT data consists of only one scene, we used frames from the TUD-Brussels outdoor pedestrian dataset [123] to extract mid-level elements. We trained our model following the same experimental design in Kitani et al. We use 3-fold cross validation, use a 15-step window for the MHD, and the full pixel grid

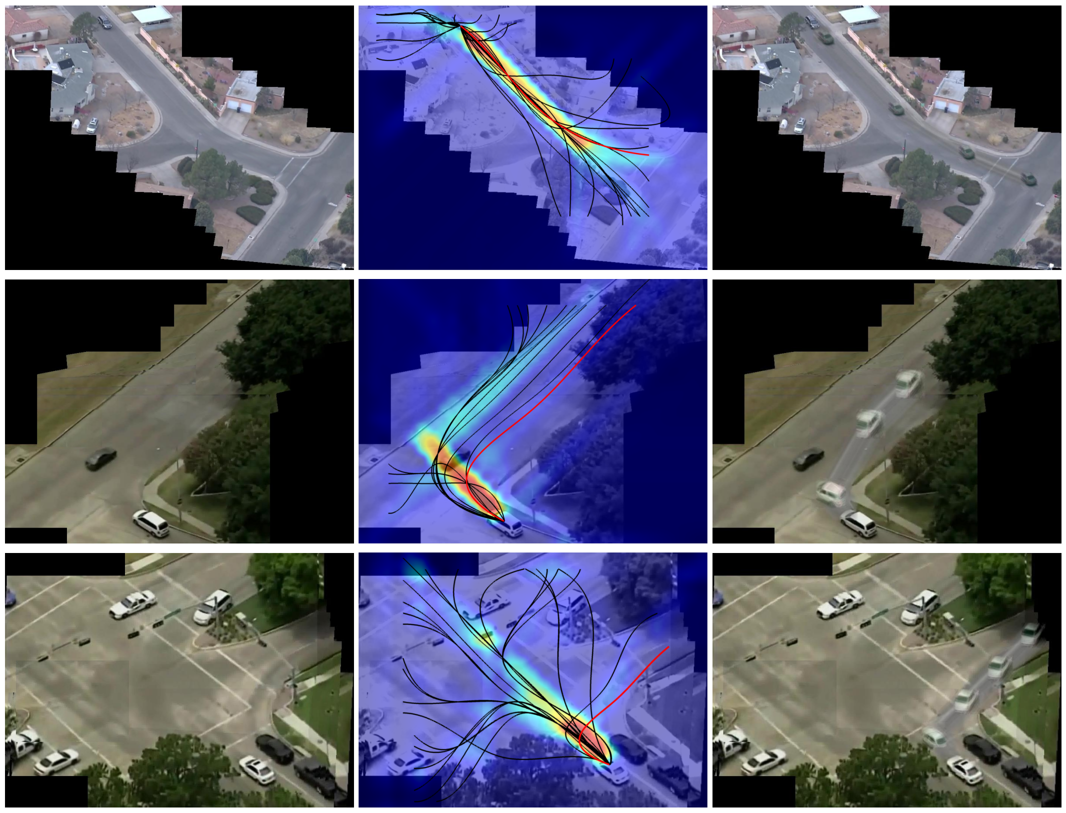

(a) Original Image (b) Prediction Heatmap (c) Predicted Path

Figure 5.8: Qualitative predictions for our approach. The far left shows the original image, the center shows a heatmap of possible paths, and the right shows a visualization of one of those paths.

(359x634) as in Kitani et al. We find that our method is able to infer goals and paths on a better than IOC. To demonstrate that our reward function is meaningful, we also compare against MDP [54] but using our reward function instead of IOC. Table 5.5 shows the performance of the approach. For this experiment, since the agent is given, we compare the top trajectory predicted by each approach.

5.4 Conclusion

In this chapter we have presented a simple and effective framework for visual prediction on a static scene. Our prediction framework builds upon representative and discriminative mid-level elements and combines this visual representation with a decision theoretic framework. This representation allows us to train our framework in a completely unsupervised manner from a large collection of videos. However, more importantly, we can also predict how visual appearances will change in time and create a hallucination of the possible future. Empirically, we have shown that our unsupervised approach outperforms even supervised approaches on multiple datasets. It is important to note that this paper represents an initial step in the direction of general unsupervised prediction. We aim to predict events in far more general domains in the future. As our patch-based framework makes strong assumptions about the visual world—no depth, rigid objects—we will consider intermediate frames which are more flexible.

Chapter 6

Using Structured Human Pose For Video Forecasting

6.1 Introduction

Consider the image in Figure 6.1. Given the context of the scene and perhaps a few past frames of the video, we can infer what likely action this human will perform. This man is outside in the snow with skis. What is he going to do in the near future? We can infer he will move his body forward towards the viewer. Given this goal of forecasting, how do we proceed? How can we predict events in a data-driven way without relying on explicit semantic classes or human-labeled data? In this chapter, we take the next step and use structure as aid in video forecasting. We explicitly separate the factors of variation in forecasting. In order to forecast, we first must determine what is active in the scene. Second, we then need to understand how the structure of the active object will deform and move over time. Finally, we need to understand how the pixels will change given the action of the object.

In Figure 6.1, we can already tell what is active in this scene, the skier, and given a description of the man's motion, we can give a good guess as to how that motion will play out at the pixel level. He is wearing dark pants and a red coat, so we would expect the colors of his figure to still be fairly coherent throughout the motion. However, the way he skis forward is fairly uncertain. He is moving towards the viewer, but he might move to the left or right as he proceeds. Models that either try to directly forecast pixels [76,90,102,111,125]

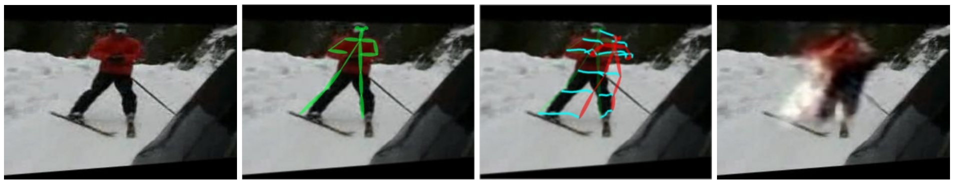

(a) Input Image (b) Detected Pose (c) Predicted Pose (d) Predicted Pixels

Figure 6.1: In this chapter, we train a generative model that takes in (a) an initial clip with (b) a detected pose. Given this information, we generate different motions in (c) pose space using a variational autoencoder and utilize a generative adversarial network to generate (d) pixels of the forecast video. Best seen in video.

or pixel motion [28,87,113,114,128] are forced to perform all of these tasks simultaneously. What makes the problem harder for a complete end-to-end approach is that it has to simultaneously learn the underlying structure—what pixels move together, the underlying physics and dynamics—how the pixels move—and the underlying low-level rendering factors such as illumination. Forecasting models may instead benefit if they explicitly separate the structure of objects from their low-level pixel appearance.

The most common agent in videos is a human. In terms of obtaining the underlying structure, there have been major advances in human pose estimation [8,11,14,121] in images, making 2D human pose a viable "free" signal in video. In this chapter, we exploit these advances to self-label video and aid forecasting. We propose a new approach to video forecasting by leveraging a more tractable space—human pose—as intermediate representation. Finally, we combine the strengths of VAE with those of GANs. The VAE estimates the probability distribution over future poses given a few initial frames. We can then forecast different plausible events in pose space. Given this structure, we then can use a generative adversarial network to fill in the details and map to pixels, generating a full video. Our approach does not rely on any explicit class labels, human labeling, or any prior semantic information beyond the presence of humans. We provide experimental results that show our model is able to account for the uncertainty in forecasting and generate plausible videos.

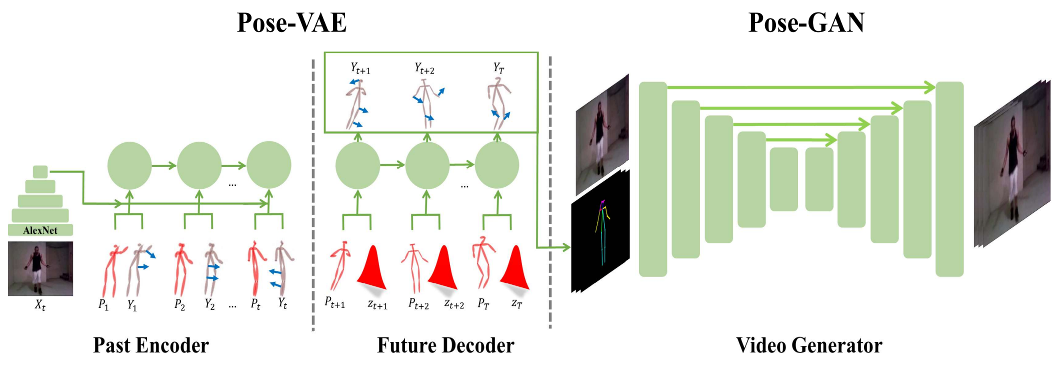

Pose-VAE
Pose-GAN
AlexNet
X_t
P_1 Y_1 P_2 Y_2 ... P_t Y_t
Y_{t+1} Y_{t+2} Y_T
P_{t+1} z_{t+1} P_{t+2} z_{t+2} P_T z_T
Past Encoder
Future Decoder
Video Generator

Figure 6.2: Overview of our approach. We use an LSTM, the Past Encoder, to encode the past input into a hidden state. We then input this hidden state into an LSTM with a Variational Autoencoder, the Future Decoder, which predicts future pose velocities based on random samples from latent variables. Given a rendered video of a pose video, we feed this with the input clip into an adversarially trained generator to output the final future video.

6.2 Methodology

In this chapter we break down video forecasting into two steps. We first predict the high-level movement in pose space using the **Pose-VAE**. Then we use this structure to predict a final pixel level video with the **Pose-GAN**.

6.2.1 Pose-VAE

The first step in our pipeline is forecasting in pure pose space. At time t, given a series of past poses $P_{1..t}$ and the last frame of in input video X_t, we want to predict the future poses up to time step T, $P_{t+1..T}$. $P_t \in \mathcal{R}^{36}$ is a 2D pose as timestep t represented by the (x, y) locations of 18 key-points. We actually predict a series of pose velocities $Y_{t+1..T}$. Given the pose velocities and an initial pose we can then construct the future pose sequence. To accomplish this forecasting task, we build upon ideas related to sequential encoder-decoder networks [32, 102]. As in these papers we can use an LSTM to encode the past information sequence. We call this the **Past Encoder** which takes in the past information X_t, $P_{1..t}$, and $Y_{1..t}$ and encodes it in a hidden representation H_t. We also have **Past Decoder** module to reconstruct the past information from the hidden state. Given this encoding H_t of the

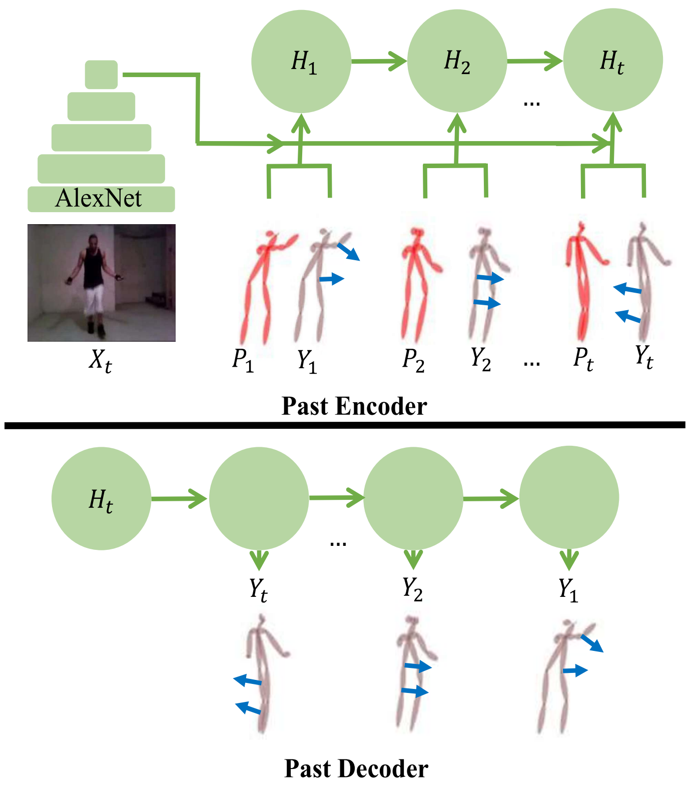


Figure 6.3: **Past Encoder-Decoder Network.** This portion of Pose-VAE encodes the past input deterministically. The Past Encoder reads in image features from X_t, corresponding past poses $P_{1..t}$, and their corresponding velocities $Y_{1...t}$. The Past Decoder replays the pose velocities in reverse order. The Past Decoder is only used for training and is discarded during testing.

past, it would be tempting to use another LSTM to simply produce the future sequence of poses similar to [102]. However, forecasting the future is not a deterministic problem; there may be multiple plausible outcomes of a video. Forecasting actually requires estimating a probability distribution over possible events. To solve this problem, we use a probabilistic Future Decoder. Our probabilistic decoder is nothing but a conditional variational autoencoder where the future velocity Y_{t+1} is predicted given the past information H_t, the current pose P_{t+1} (estimated from P_t and Y_t), and the random latent vector z_{t+1}. The hidden states of the Future Decoder are updated using the standard LSTM update rules.

Variational Autoencoders: A variational autoencoder [53] attempts to estimate the probability distribution $P(Y|z)$ of its input data Y given latent variables z. An encoder $Q(z|Y)$ learns to encode the inputs into a stochastic latent variable z. The decoder $P(Y|z)$ then reconstructs the inputs based on what is sampled from z. During training, z is regularized to match $\mathcal{N}(0,1)$ through KL-Divergence. During testing we can then sample our distribution of Y by first sampling $z \sim \mathcal{N}(0,1)$ and then feeding our sample through a neural network $P(Y|z)$ to create a sample from the distribution of Y. Another interpretation is that the decoder P transforms the latent random variable $z \sim \mathcal{N}(0,1)$ into random variable $Y \sim P(Y|z)$.

In our case, we want to estimate a distribution of future pose velocities given the past. Thus we aim to "encode" the future into latent variables $z = [z_{t+1}, z_{t+2}, ...z_T]$. Concretely, we wish to learn a way to estimate the distribution $P(Y_{t+1..T}|z, H_t)$ of future pose velocities $Y_{t+1..T}$ given our encoded knowledge of the past H_t. Thus we need to train a "Future Encoder" that learns an encoding for latent variables $z \sim Q(z|Y_{t+1..T}, H_t)$, where Q is trained to match $\mathcal{N}(0,1)$ as closely as possible. During testing, as in [113], we sample $z \sim \mathcal{N}(0,1)$ and feed sampled z values into the future decoder network to output different possible forecasts.

Past Encoder-Decoder: Figure 6.3 shows the Past Encoder-Decoder. The Past Encoder takes as input a frame X_t, a series of previous poses $P_{1..t}$, and the previous pose velocities $Y_{1..t}$. We apply a convolutional neural network on X_t. The units from the pose information and the image features are concatenated and then fed into an LSTM. After encoding the entire sequence, we use the hidden state of the LSTM at step t, H_t to condition the Future Decoder. To enforce that H_t encodes the pose velocity, the hidden state of of the encoding LSTM is fed into a decoder LSTM, the Past Decoder, which is trained through Euclidean loss to reconstruct $Y_{1..t}$ in reverse order. This enforces that the network learns a "memory" of past inputs [102]. The Past Decoder exists only as an aid for training, and at test time, only the Past Encoder is used.

Future Encoder-Decoder: Figure 6.4 shows the Future Encoder-Decoder. The Future Encoder-Decoder is composed of a VAE encoder (Future Encoder) and a VAE decoder (Future Decoder) both conditioned on past information H_t. The Future Encoder takes the future

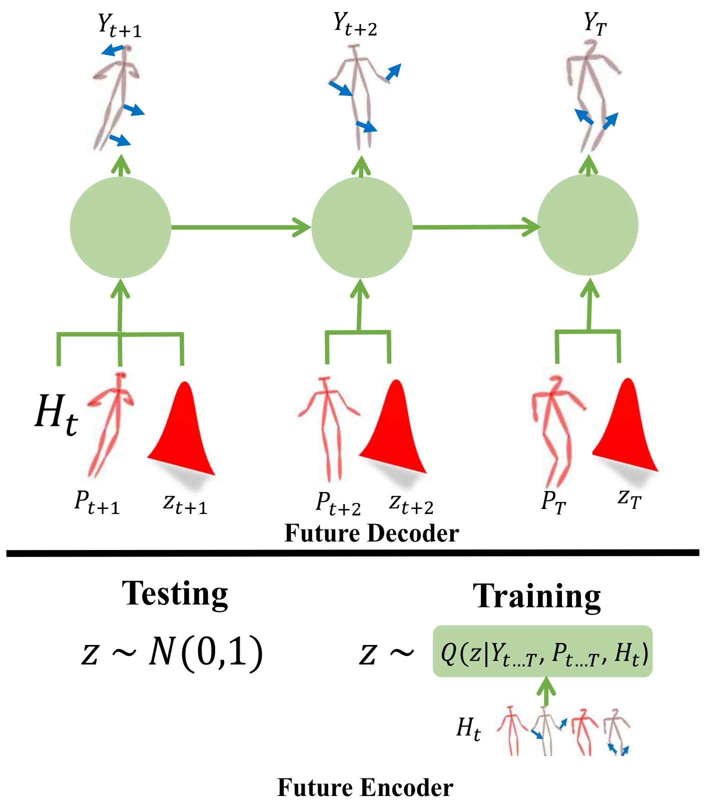


Figure 6.4: **Future Encoder-Decoder Network.** This portion of Pose-VAE encodes the future stochastically. The Future Encoder is a Variational Autoencoder which takes the past H_t and the future pose information $Y_{t+1...T}, P_{t+1...T}$ as input and outputs a Normal Distribution Q. The Future Decoder then samples z from Q to reconstruct the pose motions $Y_{t+1...T}$ given past H_t and poses $P_{t+1....T}$. During testing, the future is not known, so the Future Encoder is discarded, and only the Future Decoder is used with $z \sim \mathcal{N}(0, 1)$.

pose velocity $Y_{t+1..T}$ and the past information H_t and encodes it as a mean and variance $\mu(Y_{t+1..T}, H_t)$ and $\sigma(Y_{t+1..T}, H_t)$. We then sample a latent variable $z \sim Q(z|Y_{t+1..T}, H_t) = \mathcal{N}(\mu, \sigma)$. During testing, we sample z from a standard normal, so during training we incorporate a KL-divergence loss such that Q matches $\mathcal{N}(0, 1)$ as closely as possible. Given the latent variable z and the past information H_t, the Future Decoder recovers an approximation of the future pose sequence $\hat{Y}_{t+1..T}(z, H_t)$. The training loss for this network is the usual VAE Loss. It is Euclidean distance from the pose trajectories combined with KL-divergence loss of Q from $\mathcal{N}(0, 1)$.

$$L(\hat{Y}_{t+1..T}, Y_{t+1..T}) = ||Y_{t+1..T} - \hat{Y}_{t+1..T}||^2 + \mathcal{KL}\left[Q(z|Y_{t+1..T}, H_t)\|\mathcal{N}(0,1)\right] \tag{6.1}$$

We found in many cases the KL-term in practice can easily overwhelm the total loss, quickly reducing to the term to 0 and causing the latent variables to encode little to no useful information. In our experiments we multiply the KL-divergence loss by a constant λ in order to avoid this overregularization.

At every future step t_f, the Future Decoder takes in z_{t_f}, as well as the current pose P_{t_f} and outputs the pose motion Y_{t_f}. At training time, we use the ground truth poses, but at test time, we recover the future poses by simply adding the pose trajectory information $P_{t_f+1} = P_{t_f} + Y_{t_f}$.

Implementation Details: We train our network with Adam Solver at a learning rate of 0.001 and β_1 of 0.9. For the KL-divergence loss we set $\lambda = 0.00025$ for 60000 iterations and then set $\lambda = 0.0005$ for an additional 20000 iterations of training. Every timestep t represents 0.2 second. We conditioned the past on 2 timesteps and predict for 5 timesteps. For the convolutional network over the image network, we used an architecture almost identical to AlexNet [58] with the exception of a smaller (7x7) receptive field at the bottom layer and the addition of batch normalization layers. All layers in the entire network were trained from scratch. The LSTM units consist of two layers, both 1024 units. The Future Encoder is a simple single hidden layer network with ReLU activations and a hidden size of 512.

6.2.2 Pose-GAN

Generative Adversarial Networks: Once we sample a pose prediction from our Pose-VAE, we can then render a video of a moving skeleton. Given an input image and a video of the skeleton, we train a generative adversarial network to predict a pixel level video of future events. As described in Wang et al. [119], GANs consist of two models pitted against each other: a generator G and a discriminator D. The generator G takes the input skeleton video and image and attempts to generate a realistic video. The discriminator D, trained as a binary classifier, attempts to classify videos as either real or generated. During training, G will try to generate videos which fool D, while D will attempt to distinguish the fake videos generated by G from ones sampled from the future video frames. Following the work of [46, 111] we do not use any noise variables for the adversarial network. All the noise is contained in the Pose-VAE through z.

The loss for discriminator D is:

$$L_D = \sum_{i=1}^{M/2} l(D(V_i), l_r) + \sum_{i=M/2+1}^{M} l(D(G(I, S_T)), l_f) \tag{6.2}$$

Where V are videos, M is the batch size, I is an input image, and S_T is a video of a pose skeleton, l_r is the real label (1), and l_f is the fake label (0). Inside the batch M, half of videos V are generated, and the rest are real. The loss function l here is the binary entropy loss.

The loss for generator G is:

$$L_G = \sum_{i=M/2+1}^{M} l(D(G(I, S_T)), l_r) + \alpha ||G(I, S_T) - V_i||_1 \tag{6.3}$$

Given our Pose-VAE, we can now generate plausible pose motions given a very short clip input. For each sample, we can render a video of a skeleton visualizing how a human will deform over the last frame of the input image. Recent work in adversarial networks has shown that GANs benefit from given structure [46,84,91]. In particular, Reed et al. [91] showed that GANs improve on generating humans when given initial keypoints. In this chapter we build on this work by extending this idea to Conditional Video GANs. Given an image and a generated skeleton video, we train a GAN to generate a realistic video at the pixel level. Figure 6.5 shows the Pose-GAN network. The architecture of the discriminator D is nearly identical to that of [111].

Implementation Details: The Pose-GAN consists of five volumetric convolutional layers

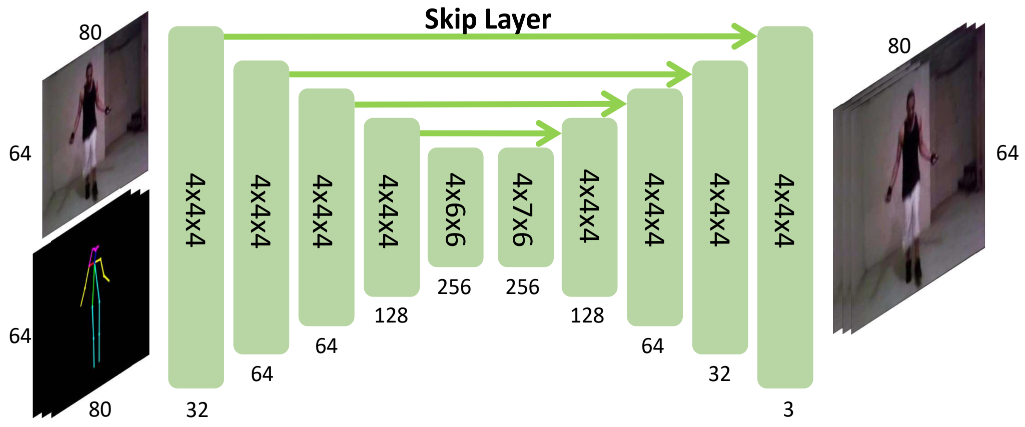


Figure 6.5: **Generator Architecture**. We use volumetric convolutions at each layer. Receptive field size represents time, width, and length. For each frame in the input pose video we stack the input frame as an extra 3 channels, making each input frame 80x64x6. The number of input and output frames is 32. The output consists of 32 frames, 80x64 pixels.

with receptive fields of 4, stride of 2, and padding of 1. At each layer LeakyReLU units and Batch Normalization are used. The only difference is that the input is a 64x80 video. For the generator G, we first encode the input using a series of five Volumentric Convolutional Layers with receptive fields of 4, stride of 2, and padding of 1. We use LeakyReLU and Batch Normalization at each layer. In order to handle the modified aspect ratio of the input (80x64), the fifth layer has a receptive field of 6 in the spatial dimensions. The top five layers are the same but in reverse, gradually increasing the spatial and temporal resolution to 64x80 pixels at 32 frames. Our training parameters are identical to [111], except that we set our regularization parameter $\alpha = 1000$. Similar to [46], we utilize skip layers for the top part of the network. For the top five layers, ReLU activation and Batch Normalization is used. The final layer is sent through a TanH function in order to scale the outputs.

6.3 Experiments

We evaluate our model on UCF-101 [101] as well as the Penn Action Dataset [131] in both pose space and video space. For the UCF-101, we utilized the training split described in [114] which uses a large portion for training data. In total we use around 1500 one-second clips for testing. To label the data in UCF-101 we utilize the pose detector of Cao et al. [8] and use the videos above an average confidence threshold. We perform temporal smoothing over the pose detections as a post-processing step. For the Penn Action Dataset, we use the standard training split.

6.3.1 Pose Evaluation

First we evaluate how well our Pose-VAE is able to forecast actions in pose space. There has been some prior work [32,47] on forecasting pose in mocap datasets such as the H3.6M dataset [45]. However, to the best of our knowledge there has been no evaluation on 2D pose forecasting on unconstrained, realistic video datasets such as UCF-101. We compare our Pose-VAE against state-of-the-art baselines. First, we study the effects of removing the VAE from our Future Decoder. In that case, the forecasting model becomes a Encoder-Recurrent-Decoder network similar to [32]. We also implemented a deterministic Structured RNN model [47] for forecasting with LSTMs explictly modeling arms, legs and torso. Finally, we take a feed-forward VAE [113] and apply it to pose trajectory prediction. In our case, the feed-forward VAE is conditioned on the image and past pose information, and it only predicts pose trajectories.

Quantitative Evaluations: For evaluation of pose forecasting, we utilize Euclidean distance from the ground-truth pose velocities. However, specifically taking the Euclidean distance over all the samples from our model in a given clip may not be very informative. Instead, we follow the evaluation proposed by Walker et al. [113]. For a set number of samples n, we see what is the best possible prediction made by the model and consider the error of closest sample from the ground-truth. We then measure how this minimum error changes as the sample size n increases and the model is given more chances. One may also consider using the highest likelihood prediction for evaluation. However, while VAEs give a distribution we can sample from, we do not know explicitly know the density function of this distribution to easily extract modes. In our qualitative results we use kmeans to approximate the mode of the distribution. A cluster with many samples means that sampling points in that cluster is likely, implying that there is a mode in that neighborhood. We show the error of the largest cluster in Figure 6.6 (Pose-VAE-Top). We make our deterministic baselines stochastic by treating the output as a mean of a multivariate normal distribution. For these baselines, we derive the bandwidth parameters from the variance of the testing data. Attempting to use the optimal MLE bandwidth via gradient search led to inferior performance. We describe the possible reasons for this phenomenon in the results section.

6.3.2 Video Evaluation

We also evaluate the final video predictions of our method. These evaluations are far more difficult as pixel space is much higher-dimensional than pose space. However, we nonetheless provide quantitative and qualitative evaluations to compare our work to the current state of the art in pixel video prediction. Specifically, we compare our method to Video-GAN [111]. For this baseline, we only make two small modifications to the original architecture—Instead of a single frame, we condition Video-GAN on 16 prior frames. We also adjust the aspect ratio of the network to output a 64x80 video.

Quantitative Evaluations: To evaluate the videos, we use the Inception score, first introduced in [97]. In the original method, the authors use the Inception model [104] to get a conditional label distribution for their generated images. In our case, we are generating videos, so we use a two-stream action classifier [116] to get a conditional label distribution $p(y|x)$ where x is our generated video and y is the action class. We calculate the label distribution by taking the average classification output of both the rgb and flow stream in the classifier. As in [97], we use the metric $\exp(\mathbb{E}_x KL(p(y|x)||p(y))$. In our case, our x is generated from an input video sequence fr and in some models a latent variable z, giving us the metric $\exp(\mathbb{E}_{fr,z} KL(p(y|x = G(f,z))||p(y))$. The intuition behind the metric is diversity; if a given classifier is highly confident of particular classes in the generated videos, then the Inception score will be large. If it has low confidence and is unsure what classes are in the videos, the conditional distribution will be close to the prior and the Inception score will be low.

We also propose a new evaluation metric based on the test statistic Maximum Mean

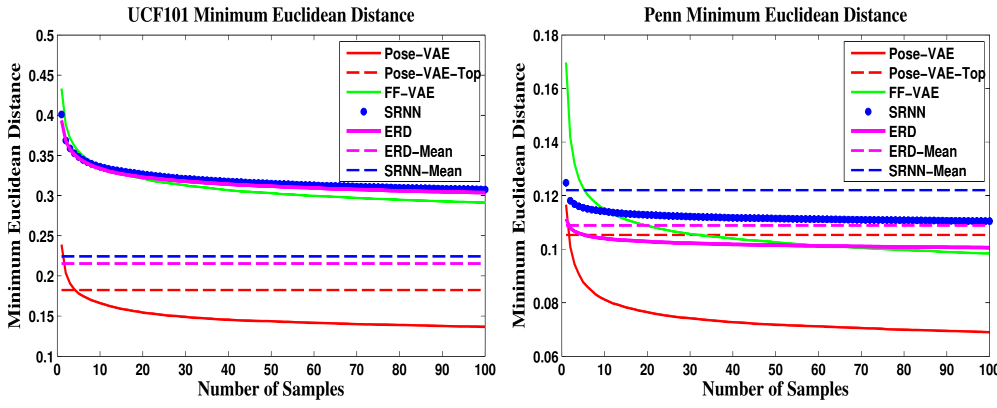


Figure 6.6: Here we show Minimum Euclidean Distance averaged over the testing examples. We take this nearest prediction in each example and plot the average of the error as the number of samples grows.

Discrepancy (MMD) [37]. MMD was proposed as a test statistic for a two sample test—given samples drawn from two distributions **P** and **Q**, we test whether or not the two distributions are equal.

While the MMD metric is based on a two sample test, and thus is a metric for how similar the generated distribution is from the ground truth, the Inception score is a rather an ad hoc metric measuring the entropy of the conditional label distribution and marginal label distribution. We present scores for both metrics, but we believe MMD to be a more statistically justifiable metric

The exact MMD statistic for a class of functions $\mathcal{F}$ is:

$$MMD[\mathcal{F}, \mathbf{P}, \mathbf{Q}] = \sup_{f \in \mathcal{F}} (\mathbb{E}_{x \sim \mathbf{P}}[f(x)] - \mathbb{E}_{y \sim \mathbf{Q}}[f(y)]). \tag{6.4}$$

Two distributions are equal if and only if for all functions $f \in \mathcal{F}$, $\mathbb{E}_x[f(x)] = \mathbb{E}_y[f(y)]$, so if $\mathbf{P} \overset{d}{=} \mathbf{Q}$, $MMD = 0$ where $\mathcal{F}$ is the set of all functions. Since evaluating over the set of all functions is intractable, we instead evaluate for all functions in a Reproducing Kernel Hilbert Space to approximate. We use the unbiased estimator for MMD from Gretton et al [37].

Some nice properties of this test statistic are that the empirical estimate is consistent

Table 6.1: Quantitative results. Higher Inception scores are better, while lower MMD scores are better.

Inception Scores

Dataset	UCF-101	Penn
Real	3.81± 0.04	3.05± 0.03
Ours	3.14± 0.04	2.73± 0.03
[111]	1.74± 0.01	2.09± 0.01

MMD Scores

Dataset	UCF-101	Penn
Real	0.003	0.006
Ours	0.022	0.009
[111]	0.139	0.060

and converges in $O(\frac{1}{\sqrt{n}})$ where n is the sample size. This is independent of the dimension of data [37]. MMD has been used in generative models before, but as part of the training procedure rather than as an evaluation criteria. Dziugaite et al. [23] uses MMD as a loss function to train a generative network to produce better images. Li et al. [65] extends this by first training an autoencoder and then training the generative network to minimize MMD in the latent space of the autoencoder, achieving less noisy images.

We choose Gaussian kernels with bandwidth ranging from 10^{-4} to 10^{9} and choose the maximum of the values generated from these bandwidths as the reported value since from eq. (6.4), we want the maximum distance out of all possible functions.

Like Inception score, we use semantic features instead of raw pixels or flow for comparison. However, we use the $fc7$ feature space rather than the labels. We concatenate the $fc7$ features from the rgb stream and the flow stream of our action classifier. This choice choice of semantic $fc7$ features is supported by the results in Li et al. [65] which show that training MMD on a lower-dimensional latent space rather than the original image space generates better looking images.

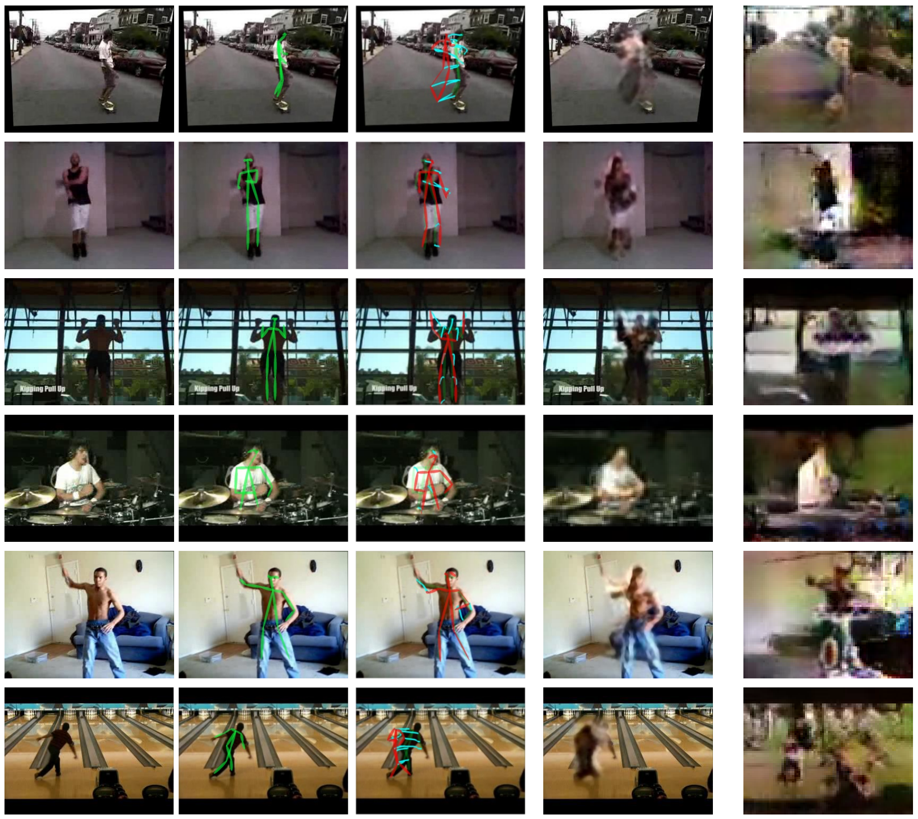

(a) Input Clip (b) Input Pose (c) Future Pose (d) Our Forecast (e) [111] Forecast

Figure 6.7: Here are some selected qualitative results from our model. Given an input clip (a) and a set of poses (b), we forecast a future pose motion (c) and then use this structure to predict video (d). These pose motions represent the largest cluster of samples from Pose-VAE for each input. Best seen in video.

6.4 Results

6.4.1 Qualitative Results

In Figure 6.7 we show the qualitative results of our model. The results are best viewed as videos. In order to generate these results, for each scene we took 1000 samples from Pose-VAE and clustered the samples above a threshold into five clusters. The pose movement shown is the largest discovered cluster. We then feed the last input frame and the future pose movement into Pose-GAN to generate the final video. On the far right we show the last predicted frame by Pose-GAN. We find that our Pose-GAN is able to forecast a plausible motion given the scene. The skateboarder moves forward, and the man in the second row, who is jumproping, moves his arms to the right side. The man doing a pullup in the third row moves his body down. The drummer plays the drums, the man in the living room moves his arm down, and the bowler recovers to standing position from his throw. We find that our Pose-GAN is able to extrapolate the pixels based on previous information. As the body deforms, the general shading and color of the person is preserved in the forecasts. We also find that Pose-GAN, to a limited extent, is able to inpaint occluded background as humans move from their starting position. In Figure 6.7 we show a side-by-side qualitative comparison of our video generation to conditional Video-GAN. While Video-GAN shows compelling results when specifically trained and tested on a specific scene category [111], we discover that this approach struggles to generate interpretable results when trained on inter-class, unconstrained videos from the UCF-101. We specifically find that [111] fails to capture even the general structure of the original input clip in many cases.

6.4.2 Quantitative Results

We show the results of our quantitative evaluation on pose prediction in Figure 6.6. We find our method is able to outperform the baselines on Euclidean distance even with a small number of samples for both datasets. The dashed lines for ERD and SRNN use the only the direct output as a mean—identical to sampling with variance 0. As expected, the Pose-VAE has a higher error with only a few samples, but as samples grow the error quickly decreases due to the stochastic nature of future pose motion. The solid lines for ERD and SRNN treat the output as a mean of a multivariate normal with variance derived from the testing data. Using the variance seems to worsen performance for these two baselines. This suggests that these deterministic baselines output one particularly incorrect motion for each of the examples, and the distribution of pose motion is not well modeled by Gaussian noise. We also find our recurrent Pose-VAE outperforms Feedforward-VAE [113]. Interestingly, FF-VAE underperforms the mean of the two deterministic baselines on UCF-101. This is likely due to the fact that FF-VAE is forced to predict all timesteps simultaneously, while recurrent models are able to predict more refined motions in a sequential manner.

In Table 6.1 we show our quantitative results of pixel-level video prediction against [111]. As the Inception score increases, the KL-Divergence between the prior distribution of labels and the conditional class label distribution given generated videos increases. Here we are effectively measuring how often the two stream action classifier detects particular classes with high confidence in the generated videos. We compute variances using bootstrapping. We find, not surprisingly, that real videos show the highest Inception score. In addition, we find that videos generated by our model have a higher Inception score than [111]. This suggests that our model is able to generate videos which are more likely to have particular meaningful features detected by the classifier. In addition to Inception scores, we show the results of our MMD metric in Table 6.1. While Inception is measuring diversity, MMD is instead testing something slightly different. Given the distribution of two sets, we perform a statistical test measuring the difference of the distributions. We find that, compared to the distribution of real videos, the distribution videos generated by [111] are much further than than ours.

6.5 Conclusion

In this chapter, we make great steps in pixel-level video prediction by exploiting pose as an essentially free source of supervision and combining the advantages of VAEs, GANs and recurrent networks. Rather than attempting to model the entire scene at once, we predict the high level dynamics of the scene by predicting the pose movements of the humans in the scenes with a VAE and then predict each pixel with a GAN. We find that our method is able to generate a distribution of plausible futures and outperform contemporary baselines. There are many future directions from this work. One possibility is to combine VAEs with the power of structured RNNs to improve performance. Another direction is to apply our model to representation learning for action recognition and early action detection; our method is unsupervised and thus could scale to large amounts of unlabeled video data.

Chapter 7

Discussion

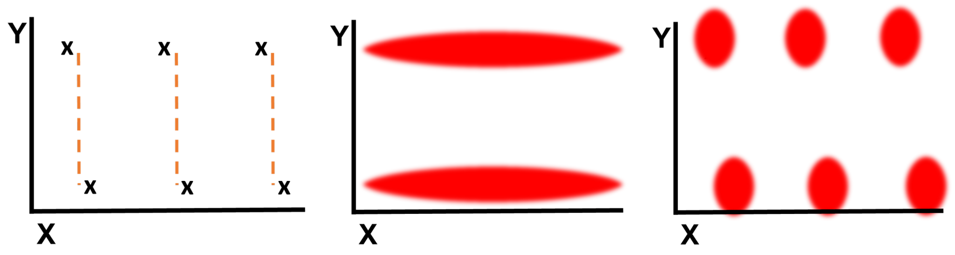
In this thesis, we explore the concept of data-driven visual forecasting. We have focused on methods of predicting events in images in a manner which is self-supervised and relies ideally on minimal semantic information. We initially focus on forecasting a fundamental aspect of visual action—simple motion information. In addition, we have taken an initial step towards prediction in structured output spaces. However, we would not say that forecasting is a completely solved problem. In this chapter we describe possible alternate directions, open problems, and limitations of current approaches.

7.1 Alternative Output Spaces

In chapters 5 and 6, we explore the use of mid-level patches and human pose as output spaces for prediction. However, there are other features that could be quite useful in other contexts. One such example is semantic segmentation—especially instance segmentation. Given that semantic segmentation performance has improved substantially in the last few years [12,41,68], this is a feature space that may be obtained in a self-labeled manner. This space preserves the concept of scene structure, object location, and object shape. However, the space of segment classes is finite and often low in number. Furthermore, this space abstracts away low-level image details such as texture, color, and shading. The work of Luc et al. [72] explores this idea, although their model focuses mostly on predicting in semantic space and does not account for multiple futures. One could combine a video prediction model in semantic mask space with a mask-to-pixel network [10,46,117] for a full pixel forecasting model. For a purely unsupervised approach, semantic masks could be replaced by masks defined by video segmentation. Another strucured space, specifically for egocentric videos, is the set of camera parameters. Assuming that two images in a video are related by a planar homography, the output space of camera motion can be described merely through a few dimensions. Finally, there is the question of whether structure can be learned directly from pixels. Instead of a predetermined feature space such as pose or semantic segments, is it possible to discover high level structure purely from images? For this idea to be a possibility, we need advances in generative models of images. Specifically, we need ways to discover the factors of variation–scene geometry, object pose, shading, texture—in the visual world and control them independently. Higher level factors in such a case would be suitable structured state spaces for forecasting. One possible foundation for this idea is PixelVAE [38] where the authors separated bedroom geometry, room configuration, and lighting in a purely-data driven manner.

7.2 Limitations of Generative Models in Forecasting

In our framework, we assume that we can model the future as a statistical distribution with a probability assigned to each possible outcome. Generative models thus seem to be useful tools for this problem. However, the distribution of the future may not be easily parameterized, and generative models implemented by neural networks may be prone to a few problems. The first is an overfit distribution to limited training data. Consider the theoretical example shown in Figure 7.1. In this example, data points sampled from a bimodal distribution may differ by a slight amount. A neural network may disentangle this difference and infer an incorrect distribution. Consider another case in a vision context. Suppose in



(a) Training Data (b) True Distribution (c) Inferred Distribution

Figure 7.1: Consider estimating the distribution from training data in (a). Suppose the true distribution (b) is bimodal, but the training samples from each mode differ from each other by a small amount on the x-axis. A generative model based on a neural network might disentangle the metric space in (a) to exaggerate these differences and infer an incorrect underlying distribution (c).

the training data we find that people with green shirts tend to move right, and people with red shirts tend to move left. A model may incorrectly latch on to shirt color as a predictor of motion when in reality this feature is not informative. The line of defense in these cases would be some form of regularization. Data augmentation through flipping and cropping images, weight initialization through pretraining, and enforcing global image consistency of individual samples are a few possible strategies. The noise in encoded latent variables in VAEs is another example [17].

The second issue is evaluation. On the most abstract level, most existing evaluations attempt to measure the probability of the ground-truth sample under the estimated distribution. This is not ideal for a few reasons. In most cases we do not have the ground-truth distribution for each testing example but only one sample. One could conceive of a number of different distributions which yield the same likelihood for the same sample. A biased estimated distribution can thus yield an overly positive evaluation. Negative log likelihood is also not as intuitive as distance metrics or accuracy percentages. One way to get around this problem is to use human judgements in the evaluation [111]; however, this can be quite expensive. How could we judge the quality of predictions in an automated fashion? This is especially difficult when we evaluate on the pixel level. How can one measure the "goodness" of pixel level results? Metrics such as the Inception score [97] and our MMD metric in chapter 6 are initial steps towards this goal.

7.3 Alternative Stochastic Models

The future is not deterministic. Instead, the future is denominated by a number of possibilities—with some more certain than others. In this thesis, we have approached this issue in a variety of ways. In Chapter 3, we simply discretize the output space of optical flow and assign each bin a probability. In Chapter 5 we also discretize by binning the action space of mid-level patches. Chapters 4 and 6 employed variational autoencoders to estimate continuous output spaces. However, there are alternative models that could be employed both in the discrete and continuous realm. For instance, the approach in Chapter 3 estimates the probability of optical flow in each pixel independently. One cannot sample multiple flows that are coherent with the rest of the image under this model. One solution would be an autoregressive layer [107] incorporating context. For models that infer low-level details such as pixels, an autoregressive network may also be useful. For example, one could easily replace the GAN in chapter 6 with a pixelCNN network, thereby making both the low-level factors of variation—pixels—stochastic with the high-level factor of human pose. Variational autoencoders also have various drawbacks. One simple alternative may be gaussian mixture models. While GMMs require an initial assumption of the type of distribution and the number of modes, they may be easier to train. They can be optimized to minimize negative log likelihood directly without the need for a regularizer such as KL-divergence. Furthermore, they can estimate and parametrize the output distribution directly. VAEs only estimate the posterior distribution implicitly and require a feedforward pass for each sample. Finally, another alternative are Bayesian neural networks [6, 29] which incorporate uncertainty in the weights of the neural network itself. This approach does not depend on training some encoded latent variable space, but still requires feedforward passes for each sample.

Chapter 8

Future Directions and Emerging Trends

Chapters 3 through 6 form the core of this thesis. Throughout this dissertation we have explored various approaches to data-driven visual forecasting. However, this topic is just one part of the larger context of higher level image understanding. Given the large recent improvements in basic recognition tasks such as object detection, classification, and spatial understanding, researchers are now starting to investigate more abstract scene information such as intuitive physics and active vision. In this chapter we now discuss future directions which build upon this thesis.

8.1 Intuitive Physics

We have largely focused on forecasting practically any event in the world. These include human actions, physical events, and the isolated decisions of agents. However, one future direction would be to specifically focus on the physical events in the world—intuitive physics. In this case, we want computers and robots to understand not only simple concepts such as falling objects and motion inertia but also more advanced physical interactions. If a wood block crashes into a block tower, where exactly will all the blocks fall? If a ball collides into another moving ball on a table, where the balls move? In order for robots especially to interact with the physical world, they must understand these concepts. Already there have been some initial excursions into this area. Work by Mottaghi et al. [77, 78] has explored the effects of Newtonian forces in static images, Wu et al. [124] have looked into inferring physical states from images, and Fragkiadaki et al. [31] forecast the dynamics of billiards balls in images.

8.2 Active Prediction

Our work has assumed that the computer is a passive observer of data. However, what if we instead assumed the computer was an agent in the visual world, capable of choosing actions which affect the state of the environment? In this sense, an agent can forecast the consequences of its own actions. Such a capability may be very useful for planning tasks and reinforcement learning. Indeed, there has already been work exploring this topic. Finn et al. [28] has considered predicting the effects of robotic pushing on objects, and Watter et al. [120] has investigated visual embeddings for control. Other papers [22, 79, 83] have looked into the realm of video games, incorporating active forecasting into reinforcement learning frameworks.

8.3 Representation Learning from Video

Forecasting events in images implicitly builds on more fundamental visual ideas. In order to predict what happens next, we first need to identify the active elements in the scene. This requires an implicit form of "active" or "moving" object detection. Next, we need to understand how these objects will change over time based on the global context in the scene. For instance, a man standing in a gym may behave differently from the same standing man in a living room. Thus a forecasting model must also learn a dictionary of distinctive visual features. It would follow that a forecasting model would learn a representation useful for other recognition tasks. We touch upon this idea very briefly in Chapter 4 with object detection, but one could apply these trained models to improve other tasks such as pose estimation, action recognition in video, or semantic segmentation.

Chapter 9

Conclusion

We have introduced approaches to data-driven visual forecasting in this dissertation. First we consider simple pixel motion and show the application of convolutional neural networks to optical flow prediction in a single image. We further generalize this idea, exploring prediction of pixel trajectories over the course of one second. Generative models such as variational autoencoders help account for the stochastic nature of forecasting. We then argue for the use of structured spaces for forecasting, abstracting away low-level features such as pixels or motion. Mid-level elements serve as one such example, and additionally we focus on human pose as another example of high to low level forecasting in video with structure. We believe that forecasting is one building block for higher-level scene understanding, related to intuitive physics, agent-based reasoning, and representation learning from video.

Bibliography

[1] P. Agrawal, J. Carreira, and J. Malik. Learning to see by moving. In *ICCV*, 2015.

[2] A. Alahi, K. Goel, V. Ramanathan, A. Robicquet, L. Fei-Fei, and S. Savarese. Social lstm: Human trajectory prediction in crowded spaces. In *CVPR*, 2016.

[3] L. Ballan, F. Castaldo, A. Alahi, F. Palmieri, and S. Savarese. Knowledge transfer for scene-specific motion prediction. In *ECCV*, 2016.

[4] M. Bar, editor. *Predictions in the Brain: Using Our Past to Generate a Future*. Oxford University Press, 2011.

[5] H. Bay, T. Tuytelaars, and L. Van Gool. Surf: Speeded up robust features. In *ECCV*, 2006.

[6] C. Blundell, J. Cornebise, K. Kavukcuoglu, and D. Wierstra. Weight uncertainty in neural networks. In *ICML*, 2015.

[7] B. Boots, A. Byravan, and D. Fox. Learning predictive models of a depth camera & manipulator from raw execution traces. In *ICRA*, 2014.

[8] Z. Cao, T. Simon, S.-E. Wei, and Y. Sheikh. Realtime multi-person 2d pose estimation using part affinity fields. In *CVPR*, 2017.

[9] Y.-W. Chao, J. Yang, B. Price, S. Cohen, and J. Deng. Forecasting human dynamics from static images. In *CVPR*, 2017.

[10] Q. Chen and V. Koltun. Photographic image synthesis with cascaded refinement networks. In *CVPR*, 2017.

[11] X. Chu, W. Yang, W. Ouyang, C. Ma, A. L. Yuille, and X. Wang. Multi-context attention for human pose estimation. In *CVPR*, 2017.

[12] J. Dai, H. Qi, Y. Xiong, Y. Li, G. Zhang, H. Hu, and Y. Wei. Deformable convolutional networks. In *ICCV*, 2017.

[13] B. De Brabandere, X. Jia, T. Tuytelaars, and L. Van Gool. Dynamic filter networks. In *NIPS*, 2016.

[14] J. Deng, Y. Kaiyu, and A. Newell. Stacked hourglass networks for human pose estimation. In *ECCV*, 2016.

[15] E. Denton, S. Chintalao, A. Szlam, and R. Fergus. Deep generative image models using a laplacian pyramid of adversarial networks. In *NIPS*, 2015.

[16] C. Doersch. *Supervision Beyond Manual Annotations for Learning Visual Representations*. PhD thesis, Carnegie Mellon University, 2016.

[17] C. Doersch. Tutorial on variational autoencoders. *arXiv preprint arXiv:1606.05908*, 2016.

[18] C. Doersch, A. Gupta, and A. A. Efros. Mid-level visual element discovery as discriminative mode seeking. In *NIPS*, 2013.

[19] C. Doersch, A. Gupta, and A. A. Efros. Unsupervised visual representation learning by context prediction. In *ICCV*, 2015.

[20] C. Doersch, S. Singh, A. Gupta, J. Sivic, and A. A. Efros. What makes Paris look like Paris? *ACM Transactions on Graphics (TOG)*, 2012.

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

[22] A. Dosovitskiy and V. Koltun. Learning to act by predicting the future. In *ICLR*, 2017.

[23] G. K. Dziugaite, D. M. Roy, and Z. Ghahramani. Training generative neural networks via maximum mean discrepancy optimization. *UAI*, 2015.

[24] D. Eigen and R. Fergus. Predicting depth, surface normals and semantic labels with a common multi-scale convolutional architecture. In *CVPR*, 2015.

[25] I. Endres, K. J. Shih, J. Jiaa, and D. Hoiem. Learning collections of part models for object recognition. In *CVPR*, 2013.

[26] C. Feichtenhofer, A. Pinz, and A. Zisserman. Convolutional two-stream network fusion for video action recognition. In *CVPR*, 2016.

[27] P. Felsen, P. Agrawal, and J. Malik. What will happen next? forecasting player moves in sports videos. In *CVPR*, 2017.

[28] C. Finn, I. Goodfellow, and S. Levine. Unsupervised learning for physical interaction through video prediction. In *NIPS*, 2016.

[29] M. Fortunato, C. Blundell, and O. Vinyals. Bayesian recurrent neural networks. *arXiv preprint arXiv:1704.02798*, 2017.

[30] D. Fouhey and C. L. Zitnick. Predicting object dynamics in scenes. In *CVPR*, 2014.

[31] K. Fragkiadaki, P. Agrawal, S. Levine, and J. Malik. Learning visual predictive models of physics for playing billiards. In *ICLR*, 2016.

[32] K. Fragkiadaki, S. Levine, P. Felsen, and J. Malik. Recurrent network models for human dynamics. In *ICCV*, 2015.

[33] R. Girshick. Fast r-cnn. In *ICCV*, 2015.

[34] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. C. Corville, and Y. Bengio. Generative adversarial nets. In *NIPS*, 2014.

[35] A. Gorban, H. Idrees, Y.-G. Jiang, A. Roshan Zamir, I. Laptev, M. Shah, and R. Sukthankar. THUMOS challenge: Action recognition with a large number of classes. http://www.thumos.info/, 2015.

[36] K. Gregor, I. Danihelka, A. Graves, D. Rezende, and D. Wierstra. Draw: A recurrent neural network for image generation. In *ICCV*, 2015.

[37] A. Gretton, K. M. Borgwardt, M. J. Rasch, B. Scholkopf, and A. Smola. A kernel two-sample test. *JMLR*, 2012.

[38] I. Gulrajani, K. Kumar, F. Ahmed, A. A. Taiga, F. Visin, D. Vazquez, and A. Courville. Pixelvae: A latent variable model for natural images. In *ICLR*, 2017.

[39] A. Gupta, P. Srinivasan, J. Shi, and L. S. Davis. Understanding videos, constructing plots learning a visually grounded storyline model from annotated videos. In *CVPR*, 2009.

[40] J. Hawkins and S. Blakeslee. *On Intelligence*. Times Books, 2004.

[41] K. He, G. Gkioxari, P. Dollár, and R. Girshick. Mask r-cnn. In *ICCV*, 2017.

[42] M. Hoai and F. De la Torre. Max-margin early event detectors. *IJCV*, 107(2):191–202, 2014.

[43] D.-A. Huang and K. M. Kitani. Action-reaction: Forecasting the dynamics of human interaction. In *ECCV*. 2014.

[44] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML*, 2015.

[45] C. Ionescu, D. Papava, V. Olaru, and C. Sminchisescu. Human3.6m: Large scale datasets and predictive methods for 3d human sensing in natural environments. *PAMI*, 2014.

[46] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros. Image-to-image translation with conditional adversarial networks. In *CVPR*, 2017.

[47] A. Jain, A. R. Zamir, S. Savarese, and A. Saxena. Structural-rnn: Deep learning on spatio-temporal graphs. In *CVPR*, 2016.

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

[49] K. Kalchbrenner, A. van den Oord, K. Simonyan, I. Dnaihelka, O. Vinyals, A. Graves, and K. Kavikcuoglu. Video pixel networks. In *ICML*, 2017.

[50] V. Karavasilis, C. Nikou, and A. Likas. Visual tracking by adaptive kalman filtering and mean shift. In *Artificial Intelligence: Theories, Models and Applications*. 2010.

[51] A. Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar, and L. Fei-Fei. Large-scale video classification with convolutional neural networks. In *CVPR*, 2014.

[52] D. P. Kingma, S. Mohamed, D. J. Rezende, and M. Welling. Semi-supervised learning with deep generative models. In *NIPS*, 2014.

[53] D. P. Kingma and M. Welling. Auto-encoding variational Bayes. *ICLR*, 2014.

[54] K. Kitani, B. Ziebart, D. Bagnell, and M. Hebert. Activity forecasting. In *ECCV*, 2012.

[55] Y. Kong, Z. Tao, and Y. Fu. Deep sequential context networks for action prediction. In *CVPR*, 2017.

[56] H. S. Koppula and A. Saxena. Anticipating human activities using object affordances for reactive robotic response. In *RSS*, 2013.

[57] P. Krähenbühl, C. Doersch, J. Donahue, and T. Darrell. Data-dependent initializations of convolutional neural networks. *ICLR*, 2016.

[58] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, 2012.

[59] H. Kuehne, H. Jhuang, E. Garrote, T. Poggio, and T. Serre. HMDB: a large video database for human motion recognition. In *ICCV*, 2011.

[60] T. Kulkarni, W. F. Whitney, P. Kohli, and J. Tenenbaum. Deep convolutional inverse graphics network. In *NIPS*, 2015.

[61] L. Ladicky, Z. Bernhard, and M. Pollefeys. Discriminatively trained dense surface normal estimation. In *ECCV*, 2014.

[62] T. Lan, T.-C. Chen, and S. Savarese. A hierarchical representation for future action prediction. In *ECCV*. 2014.

[63] I. Laptev, B. Caputo, et al. Recognizing human actions: A local svm approach. In *ICPR*, 2004.

[64] N. Lee, W. Choi, P. Vernaza, C. B. Choy, P. H. Torr, and M. Chandraker. Desire: Distant future prediction in dynamic scenes with interacting agents. In *CVPR*, 2017.

[65] Y. Li, K. Swersky, and R. Zemel. Generative moment matching networks. *JMLR*, 2015.

[66] X. Liang, L. Lee, W. Dai, and E. P. Xing. Dual motion gan for future-flow embedded video prediction. In *ICCV*, 2017.

[67] C. Liu, J. Yuen, and A. Torralba. Sift flow: Dense correspondence across scenes and its applications. *PAMI*, 2011.

[68] S. Liu, L. Qi, H. Qin, J. Shi, and J. Jia. Path aggregation network for instance segmentation. *arXiv preprint arXiv:1803.01534*, 2018.

[69] W. Lotter, G. Kreiman, and D. Cox. Deep predictive coding networks for video prediction and unsupervised learning. In *ICLR*, 2017.

[70] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *IJCV*, 60(2):91–110, 2004.

[71] C. Lu, M. Hirsch, and B. Schölkopf. Flexible spatio-temporal networks for video prediction. In *CVPR*, 2017.

[72] P. Luc, N. Neverova, C. Couprie, J. Verbeek, and Y. LeCun. Predicting deeper into the future of semantic segmentation. In *ICCV*, 2017.

[73] B. D. Lucas, T. Kanade, et al. An iterative image registration technique with an application to stereo vision. In *IJCAI*, 1981.

[74] S. Ma, L. Sigal, and S. Sclaroff. Learning activity progression in lstms for activity detection and early detection. In *CVPR*, 2016.

[75] W.-C. Ma, D.-A. Huang, N. Lee, and K. M. Kitani. Forecasting interactive dynamics of pedestrians with fictitious play. *CVPR*, 2017.

[76] M. Mathieu, C. Couprie, and Y. Lecun. Deep multi-scale video prediction beyond mean square error. In *ICLR*, 2016.

[77] R. Mottaghi, H. Bagherinezhad, M. Rastegari, and A. Farhadi. Newtonian scene understanding: Unfolding the dynamics of objects in static images. In *CVPR*, 2016.

[78] R. Mottaghi, M. Rastegari, A. Gupta, and A. Farhadi. ”what happens if...“ learning to predict the effect of forces in images. In *ECCV*, 2016.

[79] J. Oh, X. Guo, H. Lee, R. L. Lewis, and S. Singh. In *NIPS*. 2015.

[80] S. Oh, A. Hoogs, A. Perera, N. Cuntoor, C.-C. Chen, J. T. Lee, S. Mukherjee, J. Aggarwal, H. Lee, L. Davis, et al. A large-scale benchmark dataset for event recognition in surveillance video. In *CVPR*, 2011.

[81] A. Oliva and A. Torralba. Modeling the shape of the scene: A holistic representation of the spatial envelope. *IJCV*, 2001.

[82] H. S. Park, J.-J. Hwang, Y. Niu, and J. Shi. Egocentric future localization. In *CVPR*, 2016.

[83] D. Pathak, P. Agrawal, A. A. Efros, and T. Darrell. Curiosity-driven exploration by self-supervised prediction. In *ICML*, 2017.

[84] D. Pathak, P. Krahenbuhl, J. Donahue, T. Darrell, and A. A. Efros. Context encoders: Feature learning by inpainting. In *CVPR*, 2016.

[85] V. Pătrăucean, A. Handa, and R. Cipolla. Spatio-temporal video autoencoder with differentiable memory. In *ICLR Workshop*, 2016.

[86] M. Pei, Y. Jia, and S.-C. Zhu. Parsing video events with goal inference and intent prediction. In *ICCV*, 2011.

[87] S. L. Pintea, J. C. van Gemert, and A. W. Smeulders. Déjà vu: Motion prediction in static images. In *ECCV*. 2014.

[88] A. Quattoni and A. Torralba. Recognizing indoor scenes. In *CVPR*, 2009.

[89] A. Radford, L. Metz, and S. Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. 2016.

[90] M. Ranzato, A. Szlam, J. Bruna, M. Mathieu, R. Collobert, and S. Chopra. Video (language) modeling: a baseline for generative models of natural videos. *arXiv preprint arXiv:1412.6604*, 2014.

[91] S. Reed, Z. Akata, S. Mohan, S. Tenka, B. Schiele, and H. Lee. Learning what and where to draw. In *NIPS*, 2016.

[92] S. E. Reed, A. van den Oord, N. Kalchbrenner, S. Gómez, Z. Wang, D. Belov, and N. de Freitas. Parallel multiscale autoregressive density estimation. In *ICCV*, 2017.

[93] D. J. Rezende, S. Mohamed, and D. Wierstra. Stochastic backpropagation and approximate inference in deep generative models. In *ICML*, 2014.

[94] N. Rhinehart and K. M. Kitani. First-person activity forecasting with online inverse reinforcement learning. In *ICCV*, 2017.

[95] M. S. Ryoo. Human activity prediction: Early recognition of ongoing activities from streaming videos. In *ICCV*, 2011.

[96] M. S. Ryoo and L. Matthies. First-person activity recognition: Feature, temporal structure, and prediction. *IJCV*, 119(3):307–328, Sep 2016.

[97] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen. Improved techniques for training gans. In *NIPS*, 2016.

[98] T. Salimans, D. Kingma, and M. Welling. Markov chain monte carlo and variational inference: Bridging the gap. *ICML*, 2015.

[99] K. Simonyan and A. Zisserman. Two-stream convolutional networks for action recognition in videos. In *NIPS*, 2014.

[100] S. Singh, A. Gupta, and A. A. Efros. Unsupervised discovery of mid-level discriminative patches. In *ECCV*, 2012.

[101] K. Soomro, A. R. Zamir, and M. Shah. Ucf101: A dataset of 101 human actions classes from videos in the wild. *arXiv preprint arXiv:1212.0402*, 2012.

[102] N. Srivastava, E. Mansimov, and R. Salakhutdinov. Unsupervised learning of video representations using lstms. In *ICML*, 2015.

[103] S. Su, J. P. Hong, J. Shi, and H. S. Park. Predicting behaviors of basketball players from first person videos. In *CVPR*, 2017.

[104] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *CVPR*, 2015.

[105] S. D. Tran and L. S. Davis. Event modeling and recognition using markov logic networks. In *ECCV*, 2008.

[106] A. van den Oord, N. Kalchbrenner, and K. Kavukcuoglu. Pixel recurrent neural networks. In *ICML*. 2016.

[107] A. van den Oord, N. Kalchbrenner, O. Vinyals, L. Espeholt, A. Graves, and K. Kavukcuoglu. Conditional image generation with pixelcnn decoders. In *NIPS*, 2016.

[108] A. Venkatraman, N. Rhinehart, W. Sun, L. Pinto, M. Hebert, B. Boots, K. Kitani, and J. Bagnell. Predictive-state decoders: Encoding the future into recurrent networks. In *NIPS*, 2017.

[109] R. Villegas, J. Yang, Y. Zou, S. Sohn, X. Lin, and H. Lee. Learning to generate long-term future via hierarchical prediction. In *ICML*, 2017.

[110] C. Vondrick, H. Pirsiavash, and A. Torralba. Anticipating the future by watching unlabeled video. In *CVPR*, 2016.

[111] C. Vondrick, H. Pirsiavash, and A. Torralba. Generating videos with scene dynamics. In *NIPS*, 2016.

[112] C. Vondrick and A. Torralba. Generating the future with adversarial transformers. In *CVPR*, 2017.

[113] J. Walker, C. Doesrch, A. Gupta, and H. Martial. An uncertain future: Forecasting from static images using variational autoencoders. In *ECCV*, 2016.

[114] J. Walker, A. Gupta, and M. Hebert. Dense optical flow prediction from a static image. In *ICCV*, 2015.

[115] H. Wang and C. Schmid. Action recognition with improved trajectories. In *ICCV*, 2013.

[116] L. Wang, Y. Xiong, Z. Wang, and Y. Qiao. Towards good practices for very deep two-stream convnets. *arXiv preprint arXiv:1507.02159*, 2015.

[117] T.-C. Wang, M.-Y. Liu, J.-Y. Zhu, A. Tao, J. Kautz, and B. Catanzaro. High-resolution image synthesis and semantic manipulation with conditional gans. In *CVPR*, 2018.

[118] X. Wang, D. F. Fouhey, and A. Gupta. Designing deep networks for surface normal estimation. In *CVPR*, 2015.

[119] X. Wang and A. Gupta. Generative image modeling using style and structure adversarial networks. In *ECCV*, 2016.

[120] M. Watter, J. Springenberg, J. Boedecker, and M. Riedmiller. Embed to control: A locally linear latent dynamics model for control from raw images. In *NIPS*, 2015.

[121] S.-E. Wei, V. Ramakrishna, T. Kanade, and Y. Sheikh. Convolutional pose machines. In *CVPR*, 2016.

[122] P. Weinzaepfel, J. Revaud, Z. Harchaoui, and C. Schmid. DeepFlow: Large displacement optical flow with deep matching. In *ICCV*, 2013.

[123] C. Wojek, S. Walk, and B. Schiele. Multi-cue onboard pedestrian detection. In *CVPR*, 2009.

[124] J. Wu, E. Lu, P. Kohli, W. T. Freeman, and J. B. Tenenbaum. Learning to see physics via visual de-animation. In *NIPS*, 2017.

[125] T. Xue, J. Wu, K. L. Bouman, and W. T. Freeman. Visual dynamics: Probabilistic future frame synthesis via cross convolutional networks. In *NIPS*, 2016.

[126] S. Yi, H. Li, and X. Wang. Pedestrian behavior understanding and prediction with deep neural networks. In *ECCV*, 2016.

[127] Y. Yoo, K. Yun, S. Yun, J. Hong, H. Jeong, and J. Young Choi. Visual path prediction in complex scenes with crowded moving objects. In *CVPR*, 2016.

[128] J. Yuen and A. Torralba. A data-driven approach for event prediction. In *ECCV*, 2010.

[129] K.-H. Z. Zeng, W. B. Shen, D.-A. Huang, M. Sun, and J. C. Niebles. Predicting scene parsing and motion dynamics in the future. In *ICCV*, 2017.

[130] T. R. Zentall. Animals may not be stuck in time. *Learning and Motivation*, 36(2):208–225, 2005.

[131] N. Zhang, M. Paluri, M. Ranzato, T. Darrell, and L. Bourdev. Panda: Pose aligned networks for deep attribute modeling. In *CVPR*, 2014.

[132] Y. Zhou and T. L. Berg. Temporal perception and prediction in ego-centric video. In *ICCV*, 2015.

[133] Y. Zhou and T. L. Berg. Learning temporal transformations from time-lapse videos. In *ECCV*, 2016.

[134] B. D. Ziebart. *Modeling purposeful adaptive behavior with the principle of maximum causal entropy*. PhD thesis, Carnegie Mellon University, 2010.