AN ABSTRACT OF THE DISSERTATION OF

Michael Lam for the degree of Doctor of Philosophy in Computer Science presented on November 28, 2017.

Title: Fine-Grained Object Recognition Under Limited Training Data

Abstract approved: ____________________________

Sinisa Todorovic

This dissertation addresses object recognition in challenging settings, where distinct object classes are visually very similar (e.g., species of birds and insects) and/or access to training examples of object classes is limited (e.g., due to the associated high costs of data annotation). In this dissertation, we present a variety of approaches aimed at images and videos, all of which are based on our hypotheses that robust fine-grained recognition requires reasoning about discriminative image or video parts and that robust learning is possible on small training sets by disentangling factors of variations in the data. Two novel search algorithms for computer vision are presented: HC-Search and HSnet Search aimed at detecting object parts and their configurations for fine-grained recognition. In addition, the problems of video summarization and human 3D pose estimation are addressed by extending Generative Adversarial Networks (GANs) with deep recurrent networks and parameterizing GAN’s latent feature space with a Gaussian mixture model. Our theoretical and empirical results advance computer vision through demonstrated advantages of each approach relative to the state of the art.
Fine-Grained Object Recognition Under Limited Training Data

by

Michael Lam

A DISSERTATION

submitted to

Oregon State University

in partial fulfillment of
the requirements for the
degree of

Doctor of Philosophy

Presented November 28, 2017
Commencement June 2018

APPROVED:

_________________________________________________________________
Major Professor, representing Computer Science

_________________________________________________________________
Director of the School of Electrical Engineering and Computer Science

_________________________________________________________________
Dean of the Graduate School

I understand that my dissertation will become part of the permanent collection of Oregon State University libraries. My signature below authorizes release of my dissertation to any reader upon request.

_________________________________________________________________
Michael Lam, Author
ACKNOWLEDGEMENTS

I would like to thank everyone who helped make my dissertation and Ph.D. program possible. This includes my advisor Sinisa Todorovic and my Ph.D. committee. I would also like to thank my former co-advisor Thomas Dietterich for his guidance on my early research. I am grateful to my good friends and colleagues at Oregon State University who helped me with my research, directly or indirectly: Behrooz Mahasseni, Anirban Roy, Peng Lei, Dimitris Trigkakis, Shell Hu, Mohamed Amer, William Brendel, Janardhan Rao Doppa and Chao Ma. I am also very grateful for my internships at A9.com and Amazon, where I learned the practical skills to become a research engineer and scientist. Finally, I would like to thank the ARCS Portland Foundation and NSF Graduate Research Fellowship Program for helping fund my graduate studies.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th></th>
<th>Introduction</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Introduction</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td><strong>βC</strong>-Search for Structured Prediction</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>2.1 Introduction</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>2.2 Literature Review</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td>2.3 Technical Approach</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>2.3.1 <strong>βC</strong>-Search Overview</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>2.3.2 Improvements for Computer Vision</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>2.4 Experiments</td>
<td>12</td>
</tr>
<tr>
<td></td>
<td>2.4.1 Setup</td>
<td>12</td>
</tr>
<tr>
<td></td>
<td>2.4.2 Baselines</td>
<td>15</td>
</tr>
<tr>
<td></td>
<td>2.4.3 Quantitative Results</td>
<td>16</td>
</tr>
<tr>
<td></td>
<td>2.4.4 Qualitative Results</td>
<td>20</td>
</tr>
<tr>
<td></td>
<td>2.5 Summary</td>
<td>20</td>
</tr>
<tr>
<td>3</td>
<td>HSnet Search for Fine-Grained Recognition</td>
<td>24</td>
</tr>
<tr>
<td></td>
<td>3.1 Introduction</td>
<td>24</td>
</tr>
<tr>
<td></td>
<td>3.2 Literature Review</td>
<td>26</td>
</tr>
<tr>
<td></td>
<td>3.3 Technical Approach</td>
<td>27</td>
</tr>
<tr>
<td></td>
<td>3.3.1 Search Overview</td>
<td>27</td>
</tr>
<tr>
<td></td>
<td>3.3.2 Search in the Space of Image Bounding Boxes</td>
<td>28</td>
</tr>
<tr>
<td></td>
<td>3.3.3 HSnet</td>
<td>31</td>
</tr>
<tr>
<td></td>
<td>3.3.4 Learning HSnet</td>
<td>32</td>
</tr>
<tr>
<td></td>
<td>3.4 Experiments</td>
<td>33</td>
</tr>
<tr>
<td></td>
<td>3.4.1 Setup</td>
<td>33</td>
</tr>
<tr>
<td></td>
<td>3.4.2 Baselines</td>
<td>35</td>
</tr>
<tr>
<td></td>
<td>3.4.3 Quantitative Results</td>
<td>36</td>
</tr>
<tr>
<td></td>
<td>3.4.4 Qualitative Results</td>
<td>37</td>
</tr>
<tr>
<td></td>
<td>3.5 Summary</td>
<td>40</td>
</tr>
<tr>
<td>4</td>
<td>Adversarial LSTM Networks for Video Summarization</td>
<td>41</td>
</tr>
<tr>
<td></td>
<td>4.1 Introduction</td>
<td>41</td>
</tr>
<tr>
<td></td>
<td>4.2 Literature Review</td>
<td>43</td>
</tr>
<tr>
<td></td>
<td>4.3 Technical Approach</td>
<td>45</td>
</tr>
</tbody>
</table>
### TABLE OF CONTENTS (Continued)

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.3.1 Review of VAE and GAN</td>
<td>45</td>
</tr>
<tr>
<td>4.3.2 Main Components of Our Approach</td>
<td>45</td>
</tr>
<tr>
<td>4.3.3 Training of sLSTM, eLSTM, and dLSTM</td>
<td>47</td>
</tr>
<tr>
<td>4.4 Variants of our Approach</td>
<td>49</td>
</tr>
<tr>
<td>4.5 Experiments</td>
<td>51</td>
</tr>
<tr>
<td>4.5.1 Setup</td>
<td>51</td>
</tr>
<tr>
<td>4.5.2 Baselines</td>
<td>52</td>
</tr>
<tr>
<td>4.5.3 Quantitative Results</td>
<td>53</td>
</tr>
<tr>
<td>4.5.4 Qualitative Results</td>
<td>56</td>
</tr>
<tr>
<td>4.6 Summary</td>
<td>58</td>
</tr>
<tr>
<td>5 Part-Based Generative Adversarial Networks for Human 3D Pose Estimation</td>
<td>59</td>
</tr>
<tr>
<td>5.1 Introduction</td>
<td>59</td>
</tr>
<tr>
<td>5.2 Literature Review</td>
<td>61</td>
</tr>
<tr>
<td>5.3 Technical Approach</td>
<td>62</td>
</tr>
<tr>
<td>5.3.1 Models</td>
<td>62</td>
</tr>
<tr>
<td>5.3.2 GAN Generator: An Ablation Study</td>
<td>65</td>
</tr>
<tr>
<td>5.4 Experiments</td>
<td>68</td>
</tr>
<tr>
<td>5.4.1 Setup</td>
<td>68</td>
</tr>
<tr>
<td>5.4.2 Quantitative Results</td>
<td>71</td>
</tr>
<tr>
<td>5.4.3 Qualitative Results</td>
<td>73</td>
</tr>
<tr>
<td>5.5 Summary</td>
<td>74</td>
</tr>
<tr>
<td>6 Explaining Estimation of Human 3D Poses by Disentangling Generative Adversarial Networks</td>
<td>75</td>
</tr>
<tr>
<td>6.1 Introduction</td>
<td>75</td>
</tr>
<tr>
<td>6.2 Literature Review</td>
<td>77</td>
</tr>
<tr>
<td>6.3 Technical Approach</td>
<td>78</td>
</tr>
<tr>
<td>6.3.1 Generative Adversarial Network</td>
<td>79</td>
</tr>
<tr>
<td>6.3.2 Mixture of Gaussians</td>
<td>80</td>
</tr>
<tr>
<td>6.3.3 Interpretability</td>
<td>81</td>
</tr>
<tr>
<td>6.3.4 Variational Autoencoder</td>
<td>82</td>
</tr>
<tr>
<td>6.3.5 Training</td>
<td>83</td>
</tr>
<tr>
<td>6.4 Experiments</td>
<td>84</td>
</tr>
</tbody>
</table>
# TABLE OF CONTENTS (Continued)

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.4.1 Setup</td>
<td>84</td>
</tr>
<tr>
<td>6.4.2 Baselines</td>
<td>85</td>
</tr>
<tr>
<td>6.4.3 Quantitative Results</td>
<td>85</td>
</tr>
<tr>
<td>6.4.4 Qualitative Results</td>
<td>88</td>
</tr>
<tr>
<td>6.5 Summary</td>
<td>91</td>
</tr>
<tr>
<td>7 Conclusion</td>
<td>92</td>
</tr>
<tr>
<td>Bibliography</td>
<td>93</td>
</tr>
</tbody>
</table>
# LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1 Dissertation Roadmap</td>
<td>4</td>
</tr>
<tr>
<td>2.1 Overview of $\mathcal{HC}$-Search for the Semantic Scene Labeling Problem</td>
<td>8</td>
</tr>
<tr>
<td>2.2 Overview of $\mathcal{HC}$-Search Successor Function</td>
<td>11</td>
</tr>
<tr>
<td>2.3 Illustration of Relabeling Connected Components</td>
<td>12</td>
</tr>
<tr>
<td>2.4 Baselines on Nematocyst and Stanford Background Datasets</td>
<td>18</td>
</tr>
<tr>
<td>2.5 $LL$-Search, $HL$-Search, $LC$-Search and $HC$-Search</td>
<td>18</td>
</tr>
<tr>
<td>2.6 Qualitative Results on Nematocyst Dataset</td>
<td>21</td>
</tr>
<tr>
<td>2.7 Qualitative Results on Stanford Background Dataset</td>
<td>22</td>
</tr>
<tr>
<td>2.8 Qualitative Results on Make3D Dataset</td>
<td>23</td>
</tr>
<tr>
<td>3.1 Overview of HSnet</td>
<td>25</td>
</tr>
<tr>
<td>3.2 Illustration of HSnet Search Unfolded in Time</td>
<td>29</td>
</tr>
<tr>
<td>3.3 HSnet Architecture in Detail</td>
<td>30</td>
</tr>
<tr>
<td>3.4 Positions and Sizes of Bounding Boxes for Initial Search State</td>
<td>34</td>
</tr>
<tr>
<td>3.5 Plot of Average Offsets as a Function of Time Steps on CUB-2011</td>
<td>38</td>
</tr>
<tr>
<td>3.6 Sequence of Predicted Bounding Boxes for Several Images</td>
<td>39</td>
</tr>
<tr>
<td>3.7 Average Image of Cars-196 and Map Showing Clusters of Parts</td>
<td>39</td>
</tr>
<tr>
<td>4.1 Overview of Video Summarization Approach</td>
<td>42</td>
</tr>
<tr>
<td>4.2 Main Components of Video Summarization Approach</td>
<td>46</td>
</tr>
<tr>
<td>4.3 Diagram of Loss Functions Used in Training</td>
<td>47</td>
</tr>
<tr>
<td>4.4 F-score Results for Different Values of $\sigma$</td>
<td>56</td>
</tr>
<tr>
<td>4.5 Example Summaries from a Sample Video in TvSum</td>
<td>57</td>
</tr>
<tr>
<td>Figure</td>
<td>Description</td>
</tr>
<tr>
<td>--------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>5.1</td>
<td>High-Level Overview of Human 3D Pose Estimation Approach</td>
</tr>
<tr>
<td>5.2</td>
<td>Overall GAN-VAE Architecture for Human 3D Pose Estimation</td>
</tr>
<tr>
<td>5.3</td>
<td>Illustrations of Variants V1-V6</td>
</tr>
<tr>
<td>5.4</td>
<td>Qualitative 3D Pose Results from Subject S11 on Human3.6M</td>
</tr>
<tr>
<td>5.5</td>
<td>Human 3D Poses Sampled from the Latent Distribution</td>
</tr>
<tr>
<td>5.6</td>
<td>Several Interpolation Results Between Two Poses</td>
</tr>
<tr>
<td>6.1</td>
<td>Overview of Our Explainable GAN Model</td>
</tr>
<tr>
<td>6.2</td>
<td>Illustration of Our Explainable GAN Model</td>
</tr>
<tr>
<td>6.3</td>
<td>Illustration of Our Training Progression</td>
</tr>
<tr>
<td>6.4</td>
<td>Plot of Error vs. Number of Gaussian components N</td>
</tr>
<tr>
<td>6.5</td>
<td>Qualitative results on Human3.6M</td>
</tr>
<tr>
<td>6.6</td>
<td>Qualitative results on MPII</td>
</tr>
<tr>
<td>6.7</td>
<td>Skeletons Generated from Means of Gaussian Components</td>
</tr>
</tbody>
</table>
## LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Comparison of i.i.d. Edge Classifier versus Berkeley Segmentation</td>
</tr>
<tr>
<td>2.2</td>
<td>Stanford Background Dataset Results Using Segmentation Accuracy (%)</td>
</tr>
<tr>
<td>2.3</td>
<td>Make3D Dataset Results Using Relative Depth and Log-10 Depth Errors</td>
</tr>
<tr>
<td>2.4</td>
<td>Precision, Recall and F1 (%) on Nematocyst Dataset</td>
</tr>
<tr>
<td>2.5</td>
<td>Branching Factor Results</td>
</tr>
<tr>
<td>3.1</td>
<td>Comparison of State-of-the-Art Approaches on CUB-2011 Birds</td>
</tr>
<tr>
<td>3.2</td>
<td>Comparison of State-of-the-Art Approaches on Cars-196</td>
</tr>
<tr>
<td>4.1</td>
<td>Comparison of Different Variations on BenchmarkDatasets</td>
</tr>
<tr>
<td>4.2</td>
<td>Comparison of State-of-the-Art Unsupervised Approaches</td>
</tr>
<tr>
<td>4.3</td>
<td>Comparison of State-of-the-Art Supervised Approaches</td>
</tr>
<tr>
<td>4.4</td>
<td>Comparison of State-of-the-Art Approaches With Training Data Augmentation</td>
</tr>
<tr>
<td>4.5</td>
<td>Comparison of State-of-the-Art Approaches With Shallow Features</td>
</tr>
<tr>
<td>5.1</td>
<td>Comparison of State-of-the-Art Approaches on HumanEva-I (MPJPE)</td>
</tr>
<tr>
<td>5.2</td>
<td>Comparison of State-of-the-Art Approaches on Human3.6M (MPJPE)</td>
</tr>
<tr>
<td>5.3</td>
<td>Per Class Evaluation of Human3.6M by Protocol 1 (MPJPE)</td>
</tr>
<tr>
<td>5.4</td>
<td>Per Class Evaluation of Human3.6M by Protocol 2 (MPJPE)</td>
</tr>
<tr>
<td>5.5</td>
<td>Training Times for Variants on Human3.6M</td>
</tr>
<tr>
<td>6.1</td>
<td>HumanEva-I Quantitative Evaluation (MPJPE)</td>
</tr>
<tr>
<td>6.2</td>
<td>Human3.6M Per Class Evaluation (MPJPE)</td>
</tr>
<tr>
<td>6.3</td>
<td>Training Progression Baselines Study (MPJPE)</td>
</tr>
</tbody>
</table>
# LIST OF ALGORITHMS

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Greedy Heuristic Learning via DAGGER</td>
<td>13</td>
</tr>
<tr>
<td>Search-Based Fine-Grained Recognition</td>
<td>31</td>
</tr>
<tr>
<td>Training SUM/GAN Model</td>
<td>49</td>
</tr>
</tbody>
</table>
Chapter 1: Introduction

This dissertation addresses fine-grained object recognition in images and video under limited training data. One of the fundamental goals of computer vision is object recognition, the ability to automatically identify what objects are in an image or video. While this can be formulated in many ways—for instance, by drawing a bounding box around the object or labeling the relevant pixels in the image—the key challenge is to teach the computer to discriminate among distinct object classes.

What differentiates fine-grained object recognition from traditional object recognition is that in fine-grained object recognition, the distinct object classes to recognize are visually very similar, while examples in a particular object class can be visually very dissimilar. For example, certain species of birds exhibit low, subtle interclass variations (e.g., two different species of gulls look exactly the same except for the beak) while at the same time can exhibit high intraclass variations (e.g., the color and texture of a particular species differ during different times of the season). This contrasts traditional object recognition, where basic-level object classes implicitly have higher interclass variations than fine-grained classes (e.g., cat versus dog).

Fine-grained object recognition is challenging due to several reasons. As already mentioned, distinct object classes are discriminated by subtle visual differences with high intraclass variations. In addition, it is challenging to obtain labeled training data for fine-grained recognition. In a wide range of applications requiring fine-grained recognition—from military to biological images—it is either very costly (time and/or money), very difficult, or even impossible to obtain a large number of annotations. Despite these challenges, modern approaches to solving computer vision problems require a large amount of training data.

Existing approaches are not good enough for fine-grained recognition because they do not take advantage of the discriminative parts in an image and/or they require a large number of training examples. First, many modern approaches tackle fine-grained recognition by treating it as traditional object recognition and training some variant of Deep Convolutional Neural Networks (DCNN) to learn generic features [38, 121, 174]. However, as noted for fine-grained recognition, distinct object classes are discriminated by subtle visual differences. Therefore, it is advantageous to exploit this useful information in training rather than ignoring it, otherwise
training becomes unnecessarily difficult. Furthermore, limited access to available training data makes training deep networks difficult. While current approaches augment training data with outside sources [79, 161], it is very difficult (if not impossible) to obtain additional annotations in certain applications and domains.

To overcome these shortcomings, our key hypotheses are that (i) robust fine-grained recognition requires reasoning about discriminative image or video parts, and that (ii) robust learning is possible on small training sets by disentangling factors of variation in the data.

In this dissertation, we test our hypotheses by presenting a variety of approaches aimed at images and videos. In the following, we provide an outline of this dissertation (see Figure 1.1 for a visual roadmap), summarizing our methodologies and contributions.

First, we present a search-based approach for fine-grained object recognition called \textit{HC-Search} in Chapter 2 [87]. \textit{HC-Search} is a divide-and-conquer solution for structured prediction that learns multiple components with pre-defined roles (i.e. heuristic, cost, successor functions), where each of them contributes towards the overall goal by making the role of the other components easier. \textit{HC-Search} is able to reason about discriminative parts through the heuristic and cost functions. Our contribution is introducing two improvements to the \textit{HC-Search} framework for computer vision tasks: (1) a randomized segmentation space as a generic search space that leverages UCM segmentations and (2) the application of \textit{DAGGER} for training robust heuristic functions. We demonstrate that with these two improvements, \textit{HC-Search} is able to exceed the state of the art on object detection against highly-cluttered background—in other words, fine-grained recognition where foreground and background have very similar appearances. We also demonstrate state-of-the-art results on semantic scene labeling and monocular depth estimation. The results show that \textit{HC-Search}, when combined with these two improvements, is a suitable inference tool for solving not only fine-grained recognition problems but also a wide variety of computer vision structured prediction problems.

Another search-based approach, called \textit{HSnet Search}, is presented in Chapter 3 [86]. Our main contribution is a formulation of a new deep architecture called \textit{HSnet}, a deep architecture grounded by the Convolutional Neural Network (CNN) to an image and consists of three components: H-layer for computing the heuristic function, S-layer for realizing the successor function, and Long Short-Term Memory (LSTM) for capturing long-range dependencies along the search trajectory. Given an image, we use \textit{HSnet} to sequentially search for discriminative bounding boxes in the image and fuse all discovered image parts for fine-grained recognition. \textit{HSnet} provides a unified framework to jointly learn the heuristic function, which evaluates the
search states, and the successor function, which proposes new states in the search space. Our ex-
perimental results demonstrate that sequential reasoning about object parts and removing back-
ground context are effective for fine-grained recognition.

Besides search-based approaches, a Generative Adversarial Network (GAN) framework is 
presented in Chapter 4 that addresses unsupervised video summarization [96], where the task 
of picking a subset of key frames from a limited number of videos is fine-grained recognition 
of summary key frames from videos. Our key idea is to train a deep network such that the 
learned representation of the summary video and the original video should be similar. Our 
contributions include (i) a new approach to unsupervised video summarization that combines 
variational auto-encoders (VAEs) and generative-adversarial training of deep architectures, and 
(ii) the first specification of generative-adversarial training on high resolution video sequences. 
Our approach consists of summarizer and discriminator recurrent networks, where the selector 
network selects a subset of frames from the input sequence, the encoder network encodes the 
selected frames to a fixed-length feature vector and the decoder network reconstructs the video. 
A discriminator network in the GAN framework classifies the summary features as ‘original’ 
or ‘summary’ video class. Evaluation on benchmark datasets demonstrate that all unsupervised 
variations of our approach outperform the state of the art in video summarization and provides a 
comparable accuracy to the state-of-the-art supervised approaches.

Finally, Chapters 5 and 6 present approaches in the GAN framework for human 3D pose 
estimation, where predicting specific joint locations is fine-grained recognition of joints as parts 
from images. Our main framework consists of training a GAN for generating 3D human poses 
and then training a VAE to map input images to 3D poses. In Chapter 5, we present a framework 
for 3D pose estimation by combining GAN, VAE, and LSTMs. Our contributions include (i) 
modeling human 3D poses using GANs, (ii) generating 3D poses with a novel generator in GAN 
consisting of spatial and temporal LSTMs, and (iii) 3D pose estimation by combining VAE for 
encoding images with GANs that generate 3D poses. In Chapter 6, we present a similar GAN- 
VAE framework for 3D pose estimation that has the added capability to afford explanations when 
making predictions. Crucially, we introduce a mixture of Gaussians as a prior distribution in the 
latent space for disentangling factors of variation in the data, which not only helps with training 
but also provides the ability to provide explanations during inference. Our overall contributions 
include (i) extending VAE with GANs for encoding images using a mixture of Gaussians as a 
prior distribution for generating 3D poses; (ii) regularizing GANs to promote disentangling fac-
tors of variation in the latent space, and thereby affording explanations during inference; and
(iii) stabilizing GAN training with a progressive addition of outputs in the loss function. We demonstrate that while combining LSTMs with GANs and VAE in Chapter 5 achieves competitive results relative to the state of the art, introducing a mixture of Gaussians and progressive training strategy in Chapter 6 achieves superior results relative to the state of the art while also providing the additional benefit of affording explanations during inference.

This dissertation is organized in the manuscript style. Due to the diversity of prior work on each topic, we provide a complete review of the prior work on each topic at the beginning of each respective chapter. Chapter 7 closes with concluding remarks.

![Dissertation Roadmap](image)

Figure 1.1: Dissertation Roadmap
Chapter 2: \( \mathcal{HC} \)-Search for Structured Prediction

2.1 Introduction

This chapter explores whether \( \mathcal{HC} \)-Search [21] can be effective in computer vision problems, especially for fine-grained object recognition in images. \( \mathcal{HC} \)-Search has already shown the state-of-the-art performance on structured prediction problems in natural language processing. A number of tractable heuristic methods for structured prediction have been used in vision, including Dual Decomposition [84], Graph-Cuts [13], Swendsen-Wang (SW) cut [9], Variational inference [176], Quadratic Programming (QP) [118, 65], Message passing [82], a hybrid of Linear programming (LP) and QP [83], and search-based methods [44, 30, 74, 109, 126]. Instead of training a single global energy function and then solving a global optimization problem, as is done in these approaches, \( \mathcal{HC} \)-Search decomposes the problem into three steps: (Step 1) Find an initial complete solution, (Step 2) Explore a search tree of alternative candidate solutions rooted at the initial solution, and (Step 3) Score each of these candidates to select the best one. Any existing method can do Step 1. Step 2 is guided by a learned heuristic function \( \mathcal{H} \), and Step 3 is performed by a learned cost (energy) function \( \mathcal{C} \).

Doppa et al. [21] claim several advantages for \( \mathcal{HC} \)-Search. First, in the standard approaches a global energy function must be trained to “defend against” the exponentially-large set of all possible wrong answers to the problem. This is expensive both computationally and in terms of sample complexity. It can require highly expressive representations (e.g. higher-order potentials). In contrast, the heuristic function \( \mathcal{H} \) only needs to correctly rank the successors of each state that is expanded during the heuristic search in Step 2, and the cost function \( \mathcal{C} \) only needs to correctly find the best of these in Step 3. These are much easier learning problems, and hence, simpler potential functions can be applied. Second, for making predictions there is no need to solve a global optimization problem at prediction time. In effect, the system learns not only how to score candidate solutions but also how to find good candidates—it learns to do inference more efficiently. Third, \( \mathcal{HC} \)-Search can be applied to non-decomposable loss functions. This is another consequence of using a search space formulation instead of a global optimization approach. Finally, \( \mathcal{HC} \)-Search provides a clean engineering methodology for determining which
components of the system would most benefit from additional engineering effort.

Many computer vision problems can be formulated as structured prediction [132, 43, 84, 73, 6, 90, 167, 25, 138, 181, 103]. If \( \mathcal{HC} \)-Search can be applied to these, it could become a suitable inference tool for solving computer vision problems. Previous work has tested \( \mathcal{HC} \)-Search on two vision problems [85]. Results on semantic scene labeling were promising, but initial experiments on object detection against significant background clutter were disappointing. In this chapter, we claim that the shortcoming of previous work was in Step 2: the formulation of the search space and the learning algorithm employed to train \( \mathcal{H} \). This is because the \( \mathcal{HC} \)-Search specification in previous work [85] used a naive search space defined over relatively small image patches, and thus required an immense branching factor and a very deep search depth (to find a good solution). Previous work developed ad hoc search spaces that sometimes worked and sometimes did not.

In this chapter, we present an elegant and general method based on a randomized segmentation search space. The search space is defined by probabilistic sampling of a plausible image segmentation from the hierarchy of Berkeley segmentations of the image (UCM [7, 8]). This sampling is realized by randomly picking a threshold on the saliency of region boundaries present in the segmentation.

Each search step then involves changing the label of the regions defined by intersecting the chosen segmentation with connected regions defined by the current candidate solution. By choosing different thresholds for UCM, we obtain segmentations at different scales, which when intersected with the candidate solution give us search steps at multiple scales.

This search space is a big improvement, but to successfully apply \( \mathcal{HC} \)-Search, we also complement this search space by training \( \mathcal{H} \) using the advanced imitation learning algorithm DAgger [125]. The deeper search depths of computer vision problems mean that training \( \mathcal{H} \) using simple exact-imitation learning [124] is not sufficient, because exact-imitation learning does not “learn from its own mistakes.” During learning, DAgger blends the current heuristic \( \mathcal{H} \) with an oracle heuristic, which is more effective at teaching \( \mathcal{H} \) to recover from its errors.

We demonstrate that with these two improvements, \( \mathcal{HC} \)-Search is able to match or exceed the state of the art on three challenging problems: (1) semantic scene labeling, (2) monocular depth estimation, and (3) object detection against highly-cluttered background (i.e. fine-grained recognition where foreground and background have very similar appearance). We also apply the \( \mathcal{HC} \)-Search engineering methodology to evaluate the improvements introduced by the segmentation search space and DAgger training. The results show that \( \mathcal{HC} \)-Search, when combined with
these two improvements, is a suitable inference tool for solving computer vision problems.

In the following sections, Section 2.2 places HC-Search in the context of prior work; Section 2.3.1 provides an overview of HC-Search; Section 2.3.2 specifies our two improvements; and Section 2.4 presents our experiments.

2.2 Literature Review

Inference complexity in computer vision has been addressed with cascade architectures which perform multiple runs of inference from coarse to fine levels of abstraction [30, 154, 153, 105, 155]. These make inference more efficient, but they place strong restrictions on the form of the cost functions to facilitate “cascading,” and they typically require that the loss function be decomposable in a way that supports “loss augmented inference.” HC-Search places minimal restrictions on the cost function.

Classifier-based structured prediction algorithms [48, 159] make a series of local decisions towards producing a complete output \( y \) (e.g. sequential labeling of superpixels [109]). However, they typically apply the classifier in a greedy manner. This is not robust, because some greedy decisions are hard to make correctly, but they are crucial for good performance. In contrast, HC-Search searches over complete candidate solutions, which allows it to recover from errors made during greedy or local search.

Our work is also related to work on learning for inference including cost-sensitive inference [3], reinforcement learning [74, 72], and speedup learning [31]. In these paradigms, the cost function is typically known and the goal is to learn a heuristic function for directing a search algorithm to a low-cost terminal node in the search space. HC-Search learns the cost function too, and the goal is to find good solutions at any depth rather than only at well-defined terminal nodes.

Yadollahpour et al. [10, 162] proposed a re-ranking approach that is similar in spirit to HC-Search. They generate a diverse set of plausible segmentations and learn a ranking function to score them. However, their approach does not provide any guarantees for the quality of the generated outputs. In contrast, HC-Search imitates the search behavior of an oracle heuristic on training data. Based on standard learning theory considerations, it can be proved that the learned heuristic and cost function will generalize and perform well on new instances [22, 21, 125].
2.3 Technical Approach

2.3.1 HCE-Search Overview

HCE-Search [21] is a divide-and-conquer solution for structured prediction that learns multiple components with pre-defined roles, and each of them contributes towards the overall goal by making the role of the other components easier. The key elements of the HCE-Search framework include the Search space over complete outputs $S$; Search strategy $A$; Heuristic function $H: X \times Y \mapsto \mathbb{R}$ to guide the search towards high-quality outputs; and Cost function $C: X \times Y \mapsto \mathbb{R}$ to score the candidate outputs generated by the search procedure. An overview of HCE-Search is shown in Figure 2.1.

**Search Space.** Every state in $S$ consists of an input-output pair, $s = (x, y)$, representing the possibility of predicting $y$ as the output for input image $x$. Such a search space is defined in terms of two functions: (1) *Initial state function*, $I$, such that $s_0 = I(x)$ is the initial state;
and (2) *Successor function*, $S$, such that for any state $(x,y)$, $S(x,y)$ returns a set of next states $(x,y_1), \cdots, (x,y_k)$ that share the same input $x$. The initial state function can be any method that produces a complete candidate solution. In our work, $I$ will be implemented by a classifier that has been trained to predict the label of each primitive region independently.

**Search Strategy.** Guided by the heuristic function $\mathcal{H}$, the search strategy $A$ seeks high-quality candidate solutions. In this work, we will use greedy search. Greedy search traverses a path of length $\tau$ through the search space, selecting as the next state, $s_{i+1} = \arg\min_{s \in S(s_i)} H(s)$, the best successor of the current state $s_i$ according to $\mathcal{H}$.

**Making Predictions.** Given an input $x$ and time bound $\tau$, $HC$-Search first runs the search strategy $A$ guided by the heuristic $\mathcal{H}$ until the time bound is exceeded. Let $Y_{\mathcal{H}}(x)$ be the set of states visited by the search strategy $A$ within time bound $\tau$. The next step in $HC$-Search is to compute the cost $C(s)$ of each state $s \in Y_{\mathcal{H}}(x)$. The final prediction $\hat{y} = \arg\min_{s \in Y_{\mathcal{H}}(x)} C(s)$.

**Heuristic and Cost Function Learning.** To define the objective functions for learning $\mathcal{H}$ and $C$, we first decompose the overall prediction error into the errors due to $\mathcal{H}$ and $C$. The error of $HC$-Search, $\epsilon_{HC}$, for a given $\mathcal{H}$, $\tau$, and $C$ can be decomposed into two parts: 1) *Generation error*, $\epsilon_{HC}$, due to $\mathcal{H}$ not generating high-quality outputs within time limit $\tau$; and 2) *Selection error*, $\epsilon_{C|H}$, the additional error (conditional on $\mathcal{H}$) due to $C$ not selecting the best loss output generated by $\mathcal{H}$. This is formalized as

$$\epsilon_{HC} = \epsilon_{HC} + \epsilon_{C|H}.$$

, where $y^*_H$ denotes the best output that $HC$-Search could possibly return when using $\mathcal{H}$ with time limit $\tau$. Guided by the error decomposition in (2.1), the learning approach optimizes the overall error, $\epsilon_{HC}$, in a greedy stage-wise manner by first training $\mathcal{H}$ to minimize $\epsilon_{HC}$, and then, training $C$ to minimize $\epsilon_{C|H}$ conditioned on $\mathcal{H}$.

$\mathcal{H}$ is trained by imitating the search decisions made by the true loss function (only available with ground truth data). We run the search procedure $A$ for a time bound of $\tau$ for input $x$ using a heuristic equal to the true loss function, i.e. $\mathcal{H}(x,y) = L(x,y,y^*)$, and record a set of ranking constraints that are sufficient to reproduce the search behavior. For greedy search, at every search step $i$, we include one ranking constraint for every node $(x,y) \in C_i \setminus \{(x,y_{\text{best}})\}$, such that $\mathcal{H}(x,y_{\text{best}}) < \mathcal{H}(x,y)$, where $(x,y_{\text{best}})$ is the best node in the candidate set $C_i$ (ties are broken at random). The aggregate set of ranking examples is given to a rank learner (e.g. SVM-Rank) to
learn $\mathcal{H}$. This approach is called exact imitation training.

$\mathcal{C}$ is trained to score the outputs $Y_H(x)$ generated by $\mathcal{H}$ according to their true losses. Specifically, this is formulated as a bi-partite ranking problem to rank all the best loss outputs $Y_{best}$ higher than all the non-best loss outputs $Y_{H(x)} \setminus Y_{best}$.

### 2.3.2 Improvements for Computer Vision

In this section, we describe our two main improvements to $\mathcal{HC}$-Search for computer vision: (1) the randomized segmentation space and (2) training robust heuristic functions via DAGGER.

#### 2.3.2.1 The Randomized Segmentation Space

Previous work employed simple search spaces that cannot deal with the huge branching factors and search depths of computer vision problems. Lam et al. [85] employed a search space called “Flipbit.” The Flipbit space generates one successor state by changing the current label of one primitive image region (i.e. superpixel). Because the number of such regions is huge, this gives an immense branching factor, even if the number of labels is only two (hence, the name “bit”).

In our new approach, we apply the Berkeley segmentation algorithm to create an ultrametric contour map (UCM). The UCM assigns every pixel a probability (the UCM value) of belonging to an object boundary. These values have the property that for any threshold $\theta$, the regions resulting from thresholding the UCM values at $\theta$ form a set of closed regions. Regions defined by smaller thresholds are strict hierarchical refinements of the regions defined by larger thresholds. A threshold of 0 yields the finest-scale segmentation, while a threshold of 1 yields a single full-image segment.

Suppose $s_i$ is a candidate state in the search space and we wish to generate its successors $S(s_i)$. We do this as follows. First, we choose a value $\theta$ uniformly at random in $[0, 1]$ and threshold the UCM values to obtain a segmentation of the image. Second, within each segment, we define a subgraph whose nodes are the superpixels in the segment and whose edges connect adjacent superpixels. The current labeling specified by $s_i$ defines a coloring of these nodes. We compute the connected components of this subgraph, such that all nodes in a single connected component have the same color (label). For each such connected component, we generate one successor $s'$ by assigning a new label to that component.

Figure 2.2 provides an overview of our successor function. Given an input state $(x, y)$, Berke-
Figure 2.2: Overview of \( hE \)-Search Successor Function

Figure 2.3 illustrates our relabeling mechanism. After cutting all edges that fall below a threshold (gray) of the current state \((x, y)\), the graph is partitioned into subgraphs. For a particular connected component (dashed box), its label is changed to another label to create a new candidate \((x, y')\) for search. The choice of new label can be constrained (e.g., to be the label of a neighboring region.

Our approach differs from Swendsen-Wang Cuts (SW-Cut), because SW-Cut proposes merges of only pairs of neighboring image regions and proposes splits that only produce two previously-generated regions. The SW-Cut proposals are accepted based on the KL divergence of the two regions’ likelihoods. In contrast, we leverage the UCM map to propose meaningful segments. These may result in merging more than two superpixels or may split a region in a novel way, even though the whole had been assigned a single label by \( s_i \). Finally, instead of using Berkeley Segmentation, we could use any multiscale segmentation as input.
2.3.2.2 DAGGER for Training Robust Heuristics

Our second improvement is to apply DAGGER to learn robust heuristic functions when compared to the simple exact imitation training approach (see Section 2.3.1). DAGGER is an iterative algorithm that can be viewed as generating a sequence of heuristics (one per iteration), with the property that the further we go along the sequence, the greater the risk of overfitting.

Let \( \mathcal{R} \) be the set of pairwise constraints that we will use to train \( \mathcal{H} \). \( \mathcal{R} \) is initialized with the training examples generated by the exact imitation approach. Thus, the initial heuristic \( \hat{\mathcal{H}}_1 \) corresponds to the function learned from exact imitation. In all subsequent iterations \( j \), we perform search using a mixture of the learned heuristic from the previous iteration \( \hat{\mathcal{H}}_j \) and the oracle heuristic \( \mathcal{H}^* \), in other words, at each search step we make search decisions using \( \mathcal{H}^* \) (oracle) with probability \( \beta_j \) and \( \hat{\mathcal{H}}_j \) with probability \( 1 - \beta_j \), and generate additional ranking constraints whenever we make search errors. This allows DAGGER to learn from states visited by its possibly erroneous learned heuristic and correct its mistakes using oracle heuristic input. In the end, we select the heuristic that performs best on the validation data from the sequence of heuristic functions \( \mathcal{H}_1, \cdots, \mathcal{H}_{d_{\text{max}}} \). Algorithm 1 provides the pseudocode for greedy heuristic learning with DAGGER.

2.4 Experiments

2.4.1 Setup

**Datasets.** We evaluate our approach on three datasets: (1) Stanford Background Dataset [43], (2) Make3D Dataset [132, 130], and (3) Nematocyst Dataset [85]. The Stanford Background
Algorithm 1 Greedy Heuristic Learning via DAGGER

**Input:** \( \mathcal{D} = \) Training examples, \((I, S) = \) Search space definition, \(L = \) Loss function, \( \tau_{\text{max}} = \) search time bound, \( d_{\text{max}} = \) dagger iterations

**Output:** \( \mathcal{H} \), the heuristic function

1. Initialization: \( \mathcal{R} = \mathcal{R}_{\text{el}} // \) Exact imitation data
2. \( \hat{\mathcal{H}}_1 = \text{Rank-Learner}(\mathcal{R}) \)
3. for each dagger iteration \( j = 1 \) to \( d_{\text{max}} \) do
4. Current Heuristic: \( \mathcal{H}_j = \beta_j \mathcal{H}^* + (1 - \beta_j) \hat{\mathcal{H}}_j \)
5. for each training example \((x, y^*) \in \mathcal{D}\) do
6. \( s_0 = I(x) // \) initial search state
7. for each search step \( t = 1 \) to \( \tau_{\text{max}} \) do
8. Compute next states: \( C_t = S(s_{t-1}) \)
9. if \( \mathcal{H}_j(C_t) \neq \mathcal{H}^*(C_t) \) then
10. \( (x, y_{\text{best}}) = \) the best state in \( C_t \) as per \( \mathcal{H}^* \)
11. for each \((x, y) \in C_t \setminus (x, y_{\text{best}})\) do
12. Add \( \mathcal{H}(x, y_{\text{best}}) < \mathcal{H}(x, y) \) to \( \mathcal{R} \)
13. end for
14. end if
15. \( s_t = \) the best state in \( C_t \) as per \( \mathcal{H}_j \)
16. end for
17. end for
18. \( \hat{\mathcal{H}}_{j+1} = \text{Rank-Learner}(\mathcal{R}) \)
19. end for
20. return best heuristic \( \hat{\mathcal{H}}_j \) on the validation data

dataset contains 715 images of approximately 320 × 420 pixels. The ground truth assigns one of 8 semantic class labels to every pixel: sky, tree, road, grass, water, building, mountain and generic foreground object. The Make3D dataset contains 534 images of 2272 × 1704 pixels and their corresponding depth maps of resolution 55 × 305. Depth is measured in meters. The Nematocyst dataset consists of 130 grayscale images, each with a resolution of 1024 × 864 pixels. The image dataset was prepared by an expert biologist using a scanning electron microscope (SEM). The ground truth for each image is manually annotated by dividing the image into a regular grid of 32 × 32 pixel patches, and labeling each patch as belonging to the object class “basal tubule” or to “background.” The dataset is challenging due to occlusion and heavily cluttered backgrounds.

Tasks. We evaluate our new successor function and DAGGER training on scene labeling, monocular depth estimation, and object detection against clutter. For scene labeling, the task is
to assign the correct semantic class label to each pixel. For depth estimation, the task is to assign the correct real-valued depth to each pixel. For the object detection problem, the goal is to assign a label of “detected” or “not detected” to each primitive patch.

**Evaluation Setup.** For the Stanford Background dataset, we follow the five-fold cross validation experiment setup in prior work [43]. For the Make3D dataset, we employed 400 images for training and 134 images for testing as in previous work [132]. For the Nematocyst dataset, we followed the setup of previous work [85]: 80 images for training, 20 for hold-out validation, and 30 for testing.

**Metrics.** For scene labeling, our metric is accuracy, defined as the number of correctly labeled pixels divided by the total number of pixels. For depth estimation, our metrics are log-10 depth error and relative depth error. Log-10 depth error is defined as $|\log_{10} d - \log_{10} \hat{d}|$, and relative depth error is defined as $\frac{|d-\hat{d}|}{d}$, where $d$ is the ground truth depth and $\hat{d}$ is the estimated depth. For object detection, our metrics are precision, recall, and F1 measure, where true positives are $32 \times 32$ patches that fall on the ground truth tubules.

**Search Space.** Our heuristic $\mathcal{H}$ and cost $\mathcal{C}$ functions are linear in the feature function $\psi(x,y)$ that we will specify shortly: $\mathcal{H}(x,y) = w^T_h \psi(x,y)$ and $\mathcal{C}(x,y) = w^T_c \psi(x,y)$. The feature function $\psi(x,y)$ consists of unary, pairwise and context features. The unary feature $\psi_{\text{unary}}(x,y)$ is the sum of superpixel descriptors $\phi(x_i)$ for each class $1 \ldots K$: $\psi_{\text{unary}}(x,y) = \left[\sum_{i=1}^{n} I(y_i = 1)\phi(x_i), \ldots, \sum_{i=1}^{n} I(y_i = K)\phi(x_i)\right]$. In the Nematocyst dataset, each superpixel is described by a 128-dimensional SIFT descriptor. In the Stanford background dataset, each superpixel is described by 64-dimensional texture features using 64 textons, 64-dimensional color features in LAB space of 64 bins and 12-dimensional normalized grid location features, totaling 140 dimensions, as similarly used in prior work [63]. In the Make3D dataset, we used the convolutional filter features from previous work [132]. The pairwise feature $\psi_{\text{pair}}(x,y)$ captures smoothness by summing all neighboring superpixel descriptors with different labels: $\psi_{\text{pair}}(x,y) = \sum_{i,j \in \mathcal{N}} I(y_i \neq y_j)\phi(x_i) - \phi(x_j)$. The context features $\psi_{\text{context}}(x,y)$ count the $\binom{K}{2}$ co-occurrences of different labels in four different spatial relationships: “left,” “right,” “above” and “below.” Intuitively we are capturing, for example, whether a superpixel labeled “sky” is below another superpixel labeled “grass,” which the heuristic and cost functions should learn to score less favorably. Define $\psi_{\text{single-context}}(x,y,l_1,l_2,c) = \sum_{i < j} I(y_i = l_1 \land y_j = l_2 \land \text{config}(i,j,c))$ where $\text{config}(i,j,c)$ is a function that evaluates to true if superpixels $i$ and $j$ are in the configuration of $c \in \{\text{left}, \text{right}, \text{above}, \text{below}\}$. Then the context features $\psi_{\text{context}}(x,y)$ is a concatenation of $\psi_{\text{single-context}}(x,y,l_1,l_2,c)$ features over all $(l_1,l_2,c)$ combinations. The overall feature function is a concatenation of these unary, pairwise and con-
text features: $\psi(x, y) = [\psi_{\text{unary}}(x, y), \psi_{\text{pair}}(x, y), \psi_{\text{context}}(x, y)]$.

In the object detection and scene labeling tasks, we employed the Hamming loss over all pixels: $L(y, \hat{y}) = \frac{1}{n} \sum_{i=1}^{n} I(y_i \neq \hat{y}_i)$. The initial state function applies an i.i.d. classifier to all superpixels in the object detection and scene labeling settings. For the successor function, we can keep the branching factor of search reasonable by employing smarter label proposals instead of using all possible labels. In the object detection task, our successor function chooses the neighboring label for all superpixel neighbors of a connected component. Since there are only two possible labels, this has the effect of “growing” or “shrinking” detection regions of a state and not introducing “islands” of detections. In the scene labeling task, for a particular connected component $cc$, our proposal label set consists of the top $B = 3$ confident labels of $cc$ (provided by the initial state i.i.d. classifier) and the labels of the neighboring superpixels of $cc$. We found $B = 3$ to provide the best balance of accuracy versus efficiency.

In the depth estimation setting, we can treat the problem as scene labeling: each label corresponds to a depth. All depth values in the ground truth are clustered into $K = 20$ ordinal labels using K-means. A label corresponds to a depth and thus the larger the number of clusters $K$, the more accurate the prediction. We found $K = 20$ to provide the best trade-off of accuracy versus efficiency. There are several differences in the search space setup compared to scene labeling. The initial state function for depth estimation is an i.i.d. regressor that predicts a depth value for all superpixels. Each regression output is then mapped to the nearest label to yield an initial labeling for HCE-Search. We decided on the regressor over an i.i.d. classifier with 20 labels since it was significantly faster without compromising much prediction accuracy. The loss function is the log-10 depth error averaged over all pixels: $L(y, \hat{y}) = \frac{1}{n} \sum_{i=1}^{n} |\log_{10} d(y) - \log_{10} d(\hat{y})|$ where $d(y)$ is the depth value of label $y$. Each label is mapped to a depth so this loss can be computed.

Finally the successor function proposes labels as follows: for each connected component and its label $l$, propose label $l + j$ where $j = 1, -1, 5, -5$. This has the interpretation of increasing or decreasing the depth since the labels are ordinal; larger $|j|$ correspond to larger “jumps” for improving efficiency. The advantage of this setup is that despite having a large number of labels, the number of proposed labels for each connected component is not as many.

### 2.4.2 Baselines

We define the following baseline methods:

**B0. Edge Classifier:** Instead of Berkeley segmentation, one can assign weights to superpixel
graph edges similar in spirit to Swendsen-Wang Cuts. We generalize this to an i.i.d. edge classifier. Specifically, we train a logistic regression classifier on edge features and labels, where edge features are the difference of superpixel unary features on each side of the edge and labels indicate whether the edge belongs to an object boundary.

**B1. Flipbit Space:** The successor function in the Flipbit space proposes candidates by “flipping” the label of a superpixel to another label.

**B2. Exact Imitation Learning:** In exact imitation learning, the heuristic function $H$ is learned by “imitating” the search trajectory of the oracle heuristic. DAGGER refines this $H$ by training on additional examples that correct the trajectory mistakes from this $H$. This baseline does not refine $H$.

**LL/LL/LL/LL Curves.** To independently understand the quality of $H$ and $C$, we can replace either (or both) of them by an oracle that “cheats” by using the true loss function $L$ and the ground truth labels. We can generate four performance curves. $LL$ replaces both $H$ and $C$ with oracles, so it shows the best performance that can be obtained from the given search space $S$ and time limit $\tau$. $LC$ generates the best possible search candidates, but evaluates them using the learned $C$, so it tells us how good $C$ is. $HL$ uses the learned $H$ to generate the search candidates, but then evaluates them using the oracle, so it allows us to assess the learned heuristic. And of course $HC$ shows the actual performance of the system. By comparing these curves, we can understand which design choices (search space, heuristic, or cost function) would most benefit from more engineering.

**2.4.3 Quantitative Results**

We first present results on the baselines. For baseline B0, we run $LL$-Search (serves as an upper bound) with time bound $\tau = 25$ and evaluate on the appropriate metrics for the Stanford Background and Make3D datasets. We do not apply Berkeley segmentation to the Nematocyst dataset because (1) the images contain so much background clutter that Berkeley segmentation does not yield good UCMs and (2) their groundtruth data are in terms of a regular grid of patches for evaluation. Table 2.1 provides a comparison of the i.i.d. edge classifier versus Berkeley segmentation measured using $LL$-Search performance with time bound $\tau = 25$, evaluated on the Stanford Background dataset (SBD) and Make3D dataset. Table 2.1 demonstrates that using Berkeley segmentation outperforms the edge classifier. This makes sense because Berkeley segmentation’s UCM yields hierarchical closed contours, whereas the locally trained i.i.d. classifier
Table 2.1: Comparison of i.i.d. Edge Classifier versus Berkeley Segmentation

<table>
<thead>
<tr>
<th>Dataset (Metric)</th>
<th>i.i.d. Edge Classifier</th>
<th>Berkeley Segmentation</th>
</tr>
</thead>
<tbody>
<tr>
<td>SBD (Accuracy)</td>
<td>0.84</td>
<td>0.93</td>
</tr>
<tr>
<td>Make3D (Log-10 Error)</td>
<td>0.318</td>
<td>0.231</td>
</tr>
</tbody>
</table>

is not guaranteed to generate closed contours. Therefore, Berkeley segmentation is important to our search space for images of natural scenes.

Figure 2.4 compares the LL-Search performance of the B1 baseline with our approach as a function of the time bound on the Nematocyst (ND) and Stanford Background datasets (SBD). Performance for the Nematocyst dataset is the F1 score and the performance for SBD is accuracy. B1 is the LL-Search performance of Flipbit (F) versus our randomized segmentation space (RS) as a function of time bound. B2 is the $\mathcal{H}$L-Search performance of exact imitation learning (EI) versus DAGGER (DA) as a function of time bound. Since the LL-Search serves as an upper bound, it is clear that the Flipbit space would require a larger time bound to reach the same accuracy as our randomized segmentation space. In fact our search space reaches about 94% accuracy in time bound $\tau = 25$ and 94% is the upper bound on the pixel-wise accuracy given our segmentation. Thus our randomized segmentation space reduces the expected target depth of the output and speeds up our inference. Figure 2.4 also compares the performance of the B2 baseline on the Nematocyst and Stanford Background datasets. Since DAGGER seeks to improve the heuristic function, we plot the performance of $\mathcal{H}$L-Search. For DAGGER we set the parameter $\beta_i = 0.1$ and used 5 DAGGER iterations. We see DAGGER indeed improves the heuristic function over exact imitation learning. In addition, the variance of $\mathcal{H}$L-Search decreases with DAGGER, demonstrating that DAGGER learns robust heuristics.

Figure 2.5 illustrates the performance of our approach in terms of LL-Search, $\mathcal{H}$L-Search, LC-Search and HC-Search for each dataset. For the Stanford Background dataset, we see that the learned heuristic and cost functions improve upon the initial state. The $\mathcal{H}$C-Search curve demonstrates that our approach—in the absence of deeply-learned features—meets the state of the art as shown in Table 2.2. For the Make3D dataset, we see that we achieve the state-of-the-art results as shown in 2.3. Despite our loss function optimizing the log-10 depth error, our relative depth error meets close to the state of the art. For the Nematocyst dataset, we see that the learned
heuristic and cost functions perform nearly as well as \textit{LL}-Search. Indeed, in Table 2.4 we see that we exceed the state-of-the-art results on the Nematocyst dataset.

We also study the average number of candidates generated by the successor function and compare to an upper bound. We compute this for the three datasets in Table 2.5. The table provides a comparison of the number of candidates generated by the successor function averaged over all time steps and the upper bound of the branching factor. The upper bound is computed by multiplying the average number of patches or superpixels per image, \( n \), by the number of possible labels \( l \) minus 1 (the current label of a patch): \( UB = n \times (l - 1) \). This is a good upper
Table 2.2: Stanford Background Dataset Results Using Segmentation Accuracy (%)

<table>
<thead>
<tr>
<th>Method</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Region Energy [43]</td>
<td>76.4</td>
</tr>
<tr>
<td>SHL [105]</td>
<td>76.9</td>
</tr>
<tr>
<td>RNN [138]</td>
<td>78.1</td>
</tr>
<tr>
<td>ConvNet [25]</td>
<td>78.8</td>
</tr>
<tr>
<td>ConvNet + NN [25]</td>
<td>80.4</td>
</tr>
<tr>
<td>ConvNet + CRF [25]</td>
<td>81.4</td>
</tr>
<tr>
<td>Pylon (No Bnd) [90]</td>
<td>81.3</td>
</tr>
<tr>
<td>Pylon [90]</td>
<td>81.9</td>
</tr>
<tr>
<td>Ours</td>
<td>81.4</td>
</tr>
</tbody>
</table>

Table 2.3: Make3D Dataset Results Using Relative Depth and Log-10 Depth Errors

<table>
<thead>
<tr>
<th>Method</th>
<th>Relative Error</th>
<th>Log-10 Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>Learn Depth: MRF [130, 129]</td>
<td>.530</td>
<td>.198</td>
</tr>
<tr>
<td>Pointwise MRF [132, 131]</td>
<td>.458</td>
<td>.149</td>
</tr>
<tr>
<td>Superpixel MRF [132, 131]</td>
<td>.370</td>
<td>.187</td>
</tr>
<tr>
<td>Surface Layout [53]</td>
<td>1.423</td>
<td>.320</td>
</tr>
<tr>
<td>Semantic Labels [94]</td>
<td>.375</td>
<td>.148</td>
</tr>
<tr>
<td>Depth Transfer [67]</td>
<td>.361</td>
<td>.148</td>
</tr>
<tr>
<td>Ours</td>
<td>.364</td>
<td>.148</td>
</tr>
</tbody>
</table>

...bound because (1) the randomized segmentation space can select the finest segmentation, which is a collection of all patches or superpixels; and (2) the naive way of proposing labels is to consider all other possible labels. We see that the average number of candidates for all datasets is much less than the upper bound, demonstrating that our search space is more efficient.
Table 2.4: Precision, Recall and F1 (%) on Nematocyst Dataset

<table>
<thead>
<tr>
<th>Method</th>
<th>Precision</th>
<th>Recall</th>
<th>F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>CRF [85]</td>
<td>43.2</td>
<td>36.0</td>
<td>39.3</td>
</tr>
<tr>
<td>$\mathcal{HC}$-Search Flipbit [85]</td>
<td>47.2</td>
<td>54.5</td>
<td>50.6</td>
</tr>
<tr>
<td>Ours</td>
<td>83.1</td>
<td>65.1</td>
<td>72.9</td>
</tr>
</tbody>
</table>

Table 2.5: Branching Factor Results

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Avg. No. Candidates</th>
<th>Upper Bound</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nematocysts</td>
<td>126.2</td>
<td>864</td>
</tr>
<tr>
<td>SBD</td>
<td>362.7</td>
<td>3591.9</td>
</tr>
<tr>
<td>Make3D</td>
<td>1003.5</td>
<td>10497.5</td>
</tr>
</tbody>
</table>

2.4.4 Qualitative Results

Figures 2.6, 2.7, and 2.8 display qualitative results of our approach. We see that our approach yields excellent results. In the Stanford dataset, the backgrounds are mostly labeled correctly (highlighted colors correspond to semantic class labels). In the Make3D dataset, the relative depths (black indicates depth closer to the viewer than white) appear correct in general, although the depth for the sky could be closer to white (far away) than what is predicted. In the Nematocyst dataset, most of the patches (green boxes are detected patches) are detected correctly, even in the presence of heavy background clutter.

2.5 Summary

We introduced two improvements to the $\mathcal{HC}$-Search framework for vision tasks: (1) a randomized segmentation space as a generic search space that leverages UCM segmentations and (2) the application of DAGGER for training robust heuristic functions. We demonstrated that $\mathcal{HC}$-Search with these improvements gives performance comparable to or better than the state of the art across three diverse vision tasks: semantic scene labeling, monocular depth estimation, and ob-
ject detection in biological images. Our error decomposition analysis demonstrates that our improvements achieve significant performance boosts over previous attempts to apply \( HC \)-Search in computer vision.

Our investigation showed that there is still much room for improvement in the search space design for both scene labeling (oracle accuracy is 93.32\%) and depth estimation (oracle error is 0.06). The cost functions are still making many ranking errors, which suggests that there is scope for improving the cost function representation. Yadollahpour et al. [162] observed similar phenomena in their re-ranking approach. Further future work includes designing high-quality search spaces for diverse computer vision problems, learning cost functions with rich representations to improve the accuracy and efficiency, and evaluating performance on larger datasets such as PASCAL VOC datasets.
Figure 2.7: Qualitative Results on Stanford Background Dataset
Figure 2.8: Qualitative Results on Make3D Dataset
Chapter 3: HSnet Search for Fine-Grained Recognition

3.1 Introduction

This chapter addresses fine-grained object recognition using another search-based approach. Recent work has made significant progress in terms of improving accuracy on an increasing number of object classes [93, 78, 58, 161]. In contrast to general object recognition, where contextual cues are widely considered important, fine-grained recognition has been shown to benefit from identifying critical object parts and learning only off of these parts to discriminate among similar classes [11, 14, 174, 175, 20]. In this chapter, we continue this research direction by introducing and evaluating a new deep search-based framework.

It seems that our line of work stands somewhat isolated, but still rather necessary, in the context of recent advances in deep learning for various vision problems, including object tracking [144], activity recognition [95], as well as fine-grained object recognition [79, 161]. All these approaches demonstrate significant performance improvements when the initial training dataset is augmented with additional noisy data—for instance, for learning a tracker, by random sampling around ground truth trajectories, or for learning a fine-grained object detector, by downloading noisy results of Google searches on the Internet for images of fine-grained classes. Thus, the recent findings support and motivate a flurry of new research on how to obtain more training data from various multimodal sources, as this typically leads to better performance of deep methods. However, in a wide range of applications requiring fine-grained recognition, it is very difficult (if not impossible) to obtain additional ground truth or “noisy” annotations (e.g. military, biological images of fossils). To address these applications, in this chapter, we focus on how to more optimally manipulate existing data so as to extract most discriminative features and remove background for reliable fine-grained recognition.

Our approach rests on the assumption that subtle differences among very similar but distinct object classes usually take the form of differences in object parts. Thus, these parts are bound to produce the most discriminative features for fine-grained recognition. Since the objects considered are similar, it follows that the remainder of the objects’ spatial extents are shared among the classes, and thus are likely to produce confusing features for fine-grained recognition. As
the total number, locations, shapes, and often semantic meaning of these parts are not known a priori, we develop a search-based approach that

- sequentially uncovers discriminative image parts, and
- reasons over the entire search trace for recognition.

Figure 3.1 shows an overview of our approach. Given an image, we use HSnet to sequentially search for discriminative bounding boxes in the image, and fuse all uncovered image parts for fine-grained recognition. HSnet provides a unified framework to jointly learn the heuristic function, which evaluates the search states, and the successor function, which proposes new states in the search space.

An image in our approach defines a search space of the image’s bounding boxes, represented by deep features of a convolutional neural network (CNN). In this search space, we run a search algorithm, which for a given search state proposes and moves to a new state, until a time bound. A search state at a given time is defined by bounding box proposals visited until that time. The search is defined by two functions. The successor function for the current state proposes a new state in the search space. The heuristic function scores the states, in other words, all bounding box visited in the image, and in this way guides the search toward the best image parts for recognition. When the search time expires, a classifier over the last state is used for recognition.
Our main contribution is a formulation of the new deep architecture, called HSnet, for computing the above heuristic and successor functions of our sequential search in the image. HSnet is grounded via the CNN to an image, and consists of the three components: H-layer for computing the heuristic function, S-layer for realizing the successor function, and Long Short-Term Memory (LSTM) [50] for capturing long-range dependencies along the search trajectory. Thus, the role of HSnet is twofold: to evaluate bounding box candidates and propose new bounding box candidates. Since LSTM has memory, our sequential search is not greedy. That is, the LSTM’s memory enables our cumulative definition of a search state as a set of all bounding boxes visited before that state. Consequently, HSnet has a built-in robustness mechanism for handling uncertainty (e.g. occlusion, missing parts, shape deformations), as recognition does not hinge entirely on the very last set of bounding boxes uncovered when the search ends.

Evaluation on the benchmark Caltech-UCSD Birds 200-2011 and Cars-196 datasets demonstrate our competitive performance relative to the state of the art.

In the following sections, Section 3.2 places our approach in the context of prior work, Section 3.3 specifies our approach, and Section 3.4 presents our results.

3.2 Literature Review

**Fine-Grained Recognition.** There is a wide range of methods that have been developed for fine-grained object recognition [23, 165, 166, 164, 11, 14, 174, 175, 20, 93, 78]. These approaches seek to distinguish subtle differences among similar classes typically by identifying and reasoning about the layout structure of object parts present in fine-grained classes [11, 14, 174, 175, 20].

Our approach is related to existing work aimed at finding object parts using little or no supervision of parts [23, 36, 16, 58]. For example, recent work [78] combines alignment and co-segmentation to generate parts without annotations. Also, in [58], informative object parts are learned without needing part annotations by augmenting an existing CNN architecture with a differentiable spatial transformation module. In contrast to these methods, our HSnet has built-in refinement mechanism to search for increasingly more informative parts and thus improve recognition, as well as robustness mechanism against wrongly identified parts during inference.

**Object Detection.** Our work is most similar to recent object detection methods [38, 121, 119]. Object detection has been applied to fine-grained classification in prior work, where R-CNN [38, 121, 174] is trained to detect object parts. In contrast to these works, which predict object parts to classify an image in one shot, we employ sequential reasoning leveraging LSTM
to search for object parts in order to classify an image. Additionally, we cannot directly use these approaches since their object proposals are based on objectness, whereas we need object parts, and parts are not objects.

**Search.** There is a host of search-based methods in vision [44, 30, 74, 109, 126]. For example, minimizing energy of graphical models has been addressed using Monte-Carlo Markov Chain (MCMC) inference, which in turn can be viewed as a search [9]. Our approach is closely related to those methods that seek to learn the heuristic and successor functions of the search on training data, instead of using hand-coded heuristics [3, 145, 72, 126, 87]. These methods typically define the heuristic and successor functions as separate modules that are trained disjointly. In contrast, we parameterize our heuristic and successor functions such that they have the same predictor for evaluating and proposing search candidates. Moreover, we specify a unified end-to-end learning mechanism for the heuristic and successor functions.

**Attention Models.** Our approach is also similar to methods for estimating visual attention. Attention models are aimed at identifying discriminative image parts that are most responsible for recognition [160, 100, 15, 171]. While the majority of attention models focus on one bounding box or one part of an image at a time (e.g. [15]), our HSnet identifies and reasons about multiple parts of the image at a time. Our approach is closest to Jaderberg et al. [58], since their method can be interpreted as multi-attention estimation; however, we also employ sequential reasoning and search.

### 3.3 Technical Approach

#### 3.3.1 Search Overview

This section formulates our search framework. Search is defined in a space of *search states* \( s \in S \), where \( S \) may be computationally intractable or non-enumerable, as is our case. A search algorithm \( A \) is an iterative adaptive program that yields a trajectory from a given initial state \( s_0 \) to an end state \( s_\tau \): \([s_0, s_1, \ldots, s_\tau]\). \( A \) is typically guided by two functions, called the heuristic and successor functions.

Each state \( s \) can be assigned a score using the heuristic function \( H : s \mapsto \mathbb{R} \). There are many ways to define \( H \). For example, when the goal state is known, \( A^* \) search uses the heuristic function specified in terms of a distance to the goal state. Recent work seeks to learn \( H \) on training data [3, 145, 72, 126, 87]. The end state \( s_\tau \) may be reached when the search time
expires, or alternatively when the score $H(s_t)$ is greater than a threshold. The literature also presents other more sophisticated specifications for when to stop the search.

In the case of intractable $\mathcal{S}$, search requires the successor function $S$ for partially constructing the search space. $S$ “expands” a given state $s$ to its “neighbors” $\{s_1, s_2, \ldots, s_k\} \subseteq \mathcal{S}$, in other words, it constructs a finite set of new states that can be reached by search from $s$. The specifics of deciding what and when to expand are defined by a particular search algorithm. For instance, in greedy search, the neighbor with the highest $H$ score is the next one to expand for a given state.

In the next section, we formulate fine-grained recognition as search.

### 3.3.2 Search in the Space of Image Bounding Boxes

We perform search over a deep feature map, produced by a CNN, to find the most informative bounding boxes in the image for recognition. Our search space $\mathcal{S}$ is thus defined over bounding box configurations in the deep feature map. Figure 3.2 illustrates a sample search trajectory, and provides an overview of how our search-based inference works. A search state at a given time is defined by bounding box proposals visited until that time in the image. The search is guided by the heuristic $H$ and successor $S$ functions, which are unified and jointly learned using HSnet. $S$ proposes new states, and $H$ scores the states, until the time bound $\tau$. One component of HSnet is an LSTM unit whose memory fuses all candidate bounding boxes visited along the search trajectory. The soft-max layer of HSnet outputs the final recognition.

The CNN maps an image to a deep feature map, $x \in \mathbb{R}^{H \times W \times C}$, where $H$ is the height, $W$ is the width, and $C$ is the number of channels. The convolutional layers late in deep architectures have been shown to produce informative object-characteristic features [170] and thus we will use this.

The location of a bounding box $i$ is parameterized by a tuple $l^{(i)} = (x^{(i)}, y^{(i)}, w^{(i)}, h^{(i)})$ where $(x^{(i)}, y^{(i)})$ is the center and $(w^{(i)}, h^{(i)})$ is the width and height. The ranges of $l^{(i)}$ are normalized between 0 and 1. The deep features of bounding box $i$, denoted by $x^{(i)}$ (note italics), can be deterministically identified in $x$ of the entire image. A search state $s_t \in \mathcal{S}$ at time $t$ consists of $K(t)$ bounding boxes visited before $t$, $s_t = (l_t, x_t)$ where $l_t = \{l^{(i)} : i = 1, \ldots, K(t)\}$ and similarly $x_t = \{x^{(i)} : i = 1, \ldots, K(t)\}$.

Given an initial state $s_0$, our search algorithm uncovers a trajectory $[s_0, s_1, \ldots, s_\tau]$ until time bound $\tau$. Our search is guided by the heuristic $H$ and successor $S$ functions, which are unified...
Figure 3.2: Illustration of HSnet Search Unfolded in Time

and estimated by a single deep architecture, as shown in Figure 3.2. Specifically, we parameterize \( \mathcal{H} \) and \( S \) as an HSnet such that they have the same predictor for evaluating and proposing new search states. Moreover, this allows us to specify a unified end-to-end learning mechanism for \( \mathcal{H} \) and \( S \).

Given the current state \( s_t \), \( \mathcal{H} \) computes a vector of heuristic scores (for each bounding box), \( \mathcal{H}(s_t) = \phi_t \). (This can also be interpreted as heuristic features.) The heuristic scores are passed to \( S \) to propose a set of \( k \geq 1 \) additional bounding boxes, \( S(\phi_t) = [(l_1^{(1)}, x_1^{(1)}) \ldots , (l_{k}^{(k)}, x_{k}^{(k)})] \). This "expands" \( s_t \) to the next state \( s_{t+1} = (l_{t+1}, x_{t+1}) = [s_t, s(\phi_t)] \), where the number of bounding boxes considered increases to \( K(t+1) = K(t) + k \). In every search step, \( S(\phi_t) \) predicts \( k \) spatial displacements, also called offsets, of bounding boxes relative to the previous \( k \) predictions at
time $t - 1$. In our experiments, predicting offsets rather than absolute locations of bounding boxes produced better performance.

Note that as $k$ becomes larger, $\mathcal{H}$ and $\mathcal{S}$ increase the number of parameters, which in turn
become harder to robustly learn.

Our approach is summarized in Alg. 2. In the following section, we specify HSnet.

### 3.3.3 HSnet

Figure 3.3 provides a detailed diagram of the HSnet architecture. HSnet consists of H-layer, S-layer and LSTM. The CNN extracts a deep feature map $x$ from the image. The H-layer implements $\mathcal{H}$. It computes heuristic scores $\phi$ from $k$ current bounding boxes $[x^{(1)} \ldots x^{(k)}]$ (marked red). The S-layer implements $S$. It takes $\phi$, LSTM memory and locations of the bounding boxes $[l^{(1)} \ldots l^{(k)}]$ as input, and proposes $k$ spatial offsets $[o^{(1)} \ldots o^{(k)}]$ relative to $[l^{(1)} \ldots l^{(k)}]$. This is fed back via the recurrent link to define new bounding boxes in the image. After $\tau$ search steps, the soft-max layer $C$ is used for fine-grained recognition $\hat{y}$. MLP denotes a multi-layer perceptron, ROIP denotes region of interest pooling, SM is a softmax layer and R is regression.

We parameterize $\mathcal{H}$ and $S$ as HSnet. As illustrated in Figure 3.3, HSnet takes the current state as input and produces the next state. HSnet consists of three components: H-layer, S-layer and LSTM.

LSTM [50] is a recurrent neural network with a memory cell. LSTMs have been successfully used for solving a wide range of vision problems cast as sequential decision making. In our work, we use a basic 1-layer LSTM architecture [50]. Note that our definition of a cumulative search state over all bounding boxes visited is enabled by the LSTM memory.

The H-layer implements $\mathcal{H}$. The H-layer takes deep features of $k$ bounding boxes $[x^{(1)} \ldots x^{(k)}]$ proposed in the previous search step and outputs a vector of heuristic scores, $\phi$. In the R-CNN literature [38, 121], these bounding boxes are also called regions of interest (ROIs). Each ROI

---

**Algorithm 2 Search-Based Fine-Grained Recognition**

1: **Input:** Initial State $s_0$, Time Bound $\tau$
2: **Output:** Prediction $\hat{y}$
3: Timer $t := 0$
4: **while** $t < \tau$ **do**
5: Heuristic Features $\phi_t := \mathcal{H}(s_t)$
6: Next State $s_{t+1} := s_t + S(\phi_t)$
7: $t := t + 1$
8: **end while**
9: Predict $\hat{y} := C(s_\tau)$

---
is passed to a region of interest pooling layer (ROIP) to obtain a fixed-size vector representation. All the ROIs are then concatenated and passed through a multi-layer perceptron (MLP) to produce $\phi$ as output.

The $S$-layer implements $S$. As shown in Figure 3.3, as input, the $S$-layer takes $\phi$, along with the contents of LSTM memory and the locations of the $k$ bounding boxes $[l^{(1)} \ldots l^{(k)}]$. This input is passed to a multi-layer perceptron for predicting $k$ spatial offsets $[o^{(1)} \ldots o^{(k)}]$ of new bounding boxes relative to $[l^{(1)} \ldots l^{(k)}]$.

The prediction of offsets from the output of the $S$-layer is fed back via the recurrent link to define new bounding boxes in the image. After $\tau$ search steps, the soft-max layer $C$ is used to predict the fine-grained class $\hat{y}$.

Note that our complexity is lower than that of beam search (employed in much of prior work) since our $H$ and $S$ jointly processes all $k$ bounding boxes. Our complexity is also on the order of standard LSTM processing of video sequences since our $H$ and $S$ are relatively “shallow”.

3.3.4 Learning HSnet

In this work, we consider learning HSnet in two settings: (1) access to annotations of part locations is available, and (2) part annotations are not provided in the training data. In both settings, end-to-end learning of all three components in HSnet is performed using the gradient-based backpropagation through time (BPTT), commonly used for training LSTMs.

The BPTT backpropagates the standard cross entropy loss incurred on training data when the search reaches time bound $\tau$, and the soft-max output of HSnet is used for predicting class label $\hat{y}$. This classification loss is regularized by additional loss functions, defined differently for each of the above two settings.

With Part Annotations. When part annotations are available, we are able to regularize learning of HSnet to predict locations of bounding boxes such that they align better with ground truth part annotations. Specifically, we regularize learning with the Euclidean distance between a predicted bounding box and closest ground truth part. For $k$ parts, the regularization is a sum of $k$ Euclidean distances. We compute this sum at each search step $t$, and weight it with a regularization parameter $\lambda_t$. Thus, our regularized loss in this setting is defined as

$$ L = -\log p(y) + \sum_{t=1}^{\tau} \lambda_t \sum_{i=1}^{k} ||l^{(i)} - \hat{l}^{(i)}||^2, $$

(3.1)
where the first term is the cross entropy loss and the second term is regularization. In (3.1), \( y \) denotes ground truth class label, \( p(y) \) denotes the soft-max score of HSnet for the ground truth class, \( l^{(i)} \) is the ground truth location of part \( i \), \( \hat{l}_t^{(i)} \) is the location prediction of the bounding box nearest to \( l^{(i)} \) (done greedily) at search step \( t \), and \( \lambda_t \) is a regularization hyperparameter at search step \( t \). Note that our loss function only accounts for the available annotations of centers of object parts, since we do not have access to ground truth size and shape of parts.

**Without Part Annotations.** When ground truth part annotations are not provided in training data, we seek to regularize learning of HSnet to predict locations of bounding boxes such that they are visually diverse. To this end, we regularize the cross entropy loss with a term characterized by the determinantal point processes (DPP). The DPP has been widely used in learning [81]. Our regularized loss in this setting is defined as

\[
L(\hat{y}, y) = -\log p(y) - \sum_{t=1}^{T} \lambda_t \log P_t
\]

where the first term is the cross entropy loss and the second term is DPP regularization. The hyperparameters \( \lambda_t \) control the magnitude of DPP regularization. \( P_t \) is the probability of having diverse bounding boxes at search step \( t \), defined as \( P_t = \det |\Omega_k| / \det |\Omega + I| \). \( \Omega \) is a positive semi-definite kernel matrix of affinities between all possible bounding boxes, and \( \Omega_k \) denotes the restriction of \( \Omega \) to the \( k \) selected bounding boxes. The affinities are specified as inverse Euclidean distances between locations. The determinant \( \det |\Omega_k| \) quantifies the diversity of \( k \) locations. Hence, the higher the diversity, the higher \( P_t \).

Even though we have no access to part locations in this setting, we are still able to regularize the positions of bounding boxes. DPP discourages trivial solutions of learning only a single object part. Without DPP or some other training signal on the predicted part locations, it would be much more difficult to train with just a classification objective.

### 3.4 Experiments

#### 3.4.1 Setup

**Datasets.** We evaluate on CUB-2011 [151] and Cars-196 [80] datasets. CUB-2011 contains 11,788 images of 200 species of birds and is generally considered one of the most competitive datasets for fine-grained recognition. Cars-196 has 16,185 images of 196 car types. Both datasets
have a single bounding box annotation in each image (for the entire object, not each part), and CUB-2011 moreover contains rough segmentations and 15 keypoints annotated per image. We do not use the bounding box or segmentation annotations in our experiments.

**Evaluation Setup.** For CUB-2011 and Cars-196, we follow the train, validation and test splits as provided by [151, 80].

**Metrics.** Our evaluation metric is top-1 accuracy, where a correct classification is defined as when the ground truth label is present in the top 1 most confident predictions.

**Initial Search State.** Our initial search state contains $k$ bounding boxes centered at prior locations in the image. We set $k = 15$ for CUB-2011 because there are 15 bird parts available for supervision. We set $k = 10$ for Cars-196, since we empirically observed that $k = 10$ had the best tradeoff of accuracy and speed. We designed the initial state in such a way that the initial boxes are in an overlapping grid. Figure 3.4 shows an example for $k = 10$ bounding boxes, where nine boxes are arranged in a grid and the tenth box is at the center of the image. (Yellow denotes the top-left-most box out of the mentioned 9 boxes for clarity. The 10th box is in the center with larger size, denoted by orange.) We find that this performs better than random initialization, which is harder to train. We also find that it is better to cover the entire image at first with multiple bounding boxes to obtain an “overall impression” before refining the proposals to focus on parts.

**Number of Iterations.** We empirically determined that $\tau = 15$ worked best for CUB-2011 and $\tau = 10$ worked best for Cars-196. We experimented with $\tau = 1, 2, 5, 10, 15, 20, 25, 50$ and determined the best trade-off of accuracy and computation time for each dataset. It turns out that roughly setting $\tau = k$ yields the best performance. One possible explanation is that each time step can focus on one part even if multiple bounding boxes are being refined simultaneously. We also set regularization hyperparameter $\lambda_t$ to a linear schedule, with the most weight when $t = \tau$. 

![Figure 3.4: Positions and Sizes of Bounding Boxes for Initial Search State](image)
Implementation Details. For our CNN, we employed a GoogLeNet architecture with batch normalization [54] pre-trained on ImageNet [127]. Since ImageNet contains several images from our other datasets for evaluation, we removed them from training. We use Caffe [60] for extracting feature maps from images and TensorFlow [1] for implementing HSnet. The MLP (multi-layer perceptron) layer after ROIP (region of interest pooling) contains two fully connected layers of size 4096. The MLP layer after the LSTM contains one layer of size 2048. The LSTM contains 2048 hidden units. We train our framework with Adam optimizer using the default parameters.

3.4.2 Baselines

We define the following baseline methods.

B1. CNN: Given an image, a CNN directly predicts the class. We fine-tune a pre-trained model as done in Jaderberg et al. [58]

B2. CNN with ground truth bounding boxes: Given an image, a CNN produces a feature map, then a neural network predicts the class based only on the contents of $k$ bounding boxes, initialized to the ground truth part locations. This can be thought of as one time step of search with an initial state set to the ground truth parts. We want this “cheating” baseline to demonstrate that $k$ boxes are enough to classify an image with the absence of context. Note that this baseline is only available for CUB-2011, which contain annotated part locations. Since only the part locations are given and not the bounding box sizes, we empirically determined that 64 was the best size out of 16, 32, 64, 128.

B3. HSnet with one ground truth bounding box: In this baseline, HSNet accepts as input one bounding box instead of $k$ bounding boxes. HSnet is run for a fixed sequence (predetermined before search begins) of $k$ time steps, where at each time step a ground truth bounding box is provided based on the $k$ part annotations. HSnet’s proposals are not used. Additionally, our loss function only contains the classification objective since the “proposed” bounding boxes from search are already ground truth. In this “cheating” baseline, we want to show that sequential reasoning of object parts works reasonably. Note that this baseline is also only available for CUB-2011. Again, we empirically determined that 64 was the best box size.

B4. HSnet with one bounding box: This baseline is similar to B3 except that instead of using ground truth bounding boxes, the next bounding box predicted by HSnet is used. This baseline still only uses one bounding box rather than $k$ boxes. The initial box is the center box in
Table 3.1: Comparison of State-of-the-Art Approaches on CUB-2011 Birds

<table>
<thead>
<tr>
<th>Method</th>
<th>Annotations Used</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Krause 2015 [78]</td>
<td>GT+BB</td>
<td>82.8</td>
</tr>
<tr>
<td>Jaderberg 2015 [58]</td>
<td>GT</td>
<td>84.1</td>
</tr>
<tr>
<td>Xu 2015 [161]</td>
<td>GT+BB+parts+web</td>
<td>84.6</td>
</tr>
<tr>
<td>Lin 2015 [93]</td>
<td>GT+BB</td>
<td>85.1</td>
</tr>
<tr>
<td>B1</td>
<td>GT</td>
<td>82.3</td>
</tr>
<tr>
<td>B2</td>
<td>GT+parts</td>
<td>83.1</td>
</tr>
<tr>
<td>B3</td>
<td>GT+parts</td>
<td>86.2</td>
</tr>
<tr>
<td>B4</td>
<td>GT+parts</td>
<td>85.7</td>
</tr>
<tr>
<td>HSnet</td>
<td>GT+parts</td>
<td>87.5</td>
</tr>
</tbody>
</table>

Figure 3.4. For each sequence, we only focus on one part. We train on all object parts for each image. Note that using $k$ boxes results in our proposed approach.

3.4.3 Quantitative Results

Table 3.1 compares our main result and baselines with prior work on CUB-2011. Annotations used during training time are specified as follows: “GT” denotes ground truth category labels, “BB” denotes bounding box annotations, “parts” denotes part annotations and “web” denotes augmenting dataset with web data. Our results on CUB-2011 are competitive with the state of the art with about a 3 percentage points boost. Baselines B1 and B2 are comparable, which suggests that removing some context does not hurt recognition. B3 and B4 yield higher accuracies than B1 and B2, suggesting that sequential reasoning does help. B4 is slightly worse than B3 since ground truth is not available in B4. Finally, our complete framework performs better than all the baselines, which suggests that multiple proposals are better than one proposal at a time. We also believe that our approach performs better than B3 because our model observes multiple observations of different-sized bounding boxes whereas B3 only uses one bounding box with a fixed size.

Table 3.2 compares our main result and baselines with prior work on Cars-196. Annotations used during training time are specified as follows: “GT” denotes ground truth category labels,
Table 3.2: Comparison of State-of-the-Art Approaches on Cars-196

<table>
<thead>
<tr>
<th>Method</th>
<th>Annotations Used</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Deng 2013 [20]</td>
<td>GT+BB</td>
<td>63.6</td>
</tr>
<tr>
<td>Krause 2013 [80]</td>
<td>GT+BB</td>
<td>67.6</td>
</tr>
<tr>
<td>Krause 2014 [77]</td>
<td>GT</td>
<td>73.9</td>
</tr>
<tr>
<td>Lin 2015 [93]</td>
<td>GT</td>
<td>91.3</td>
</tr>
<tr>
<td>Krause 2015 [78]</td>
<td>GT+BB</td>
<td>92.6</td>
</tr>
<tr>
<td>B1</td>
<td>GT</td>
<td>88.5</td>
</tr>
<tr>
<td>B4</td>
<td>GT</td>
<td>92.2</td>
</tr>
<tr>
<td>HSnet</td>
<td>GT</td>
<td><strong>93.9</strong></td>
</tr>
</tbody>
</table>

“BB” denotes bounding box annotations and “parts” denotes part annotations. Since part annotations are not available for Cars-196, we can only perform baselines B1 and B4. Notably, B4 performs significantly better than B1, which again supports that sequential reasoning performs better than recognition with a CNN in one shot. Overall, our complete framework performs better than the baselines, and it is also competitive with the previous state of the art.

Figure 3.5 plots the average offsets of predicted part locations to ground truth part locations as a function of search time step for CUB-2011. Offset is the distance between the ground truth part location and predicted location. Different colors indicate different parts, for a total of 15 parts split across two plots. The plots show that as the time step increases, the average offsets are decreasing, indicating that our framework is learning to localize parts.

3.4.4 Qualitative Results

Figure 3.6 shows the sequence of bounding boxes predicted for a few images of birds. We show two success cases (top two rows) and one failure case (bottom row), where a success case is when the final predicted class is correct and otherwise a failure. We show for time steps $t = 5, 10, 15$, where $t = 15$ is the final time step used for classification. We also compare these bounding box locations with the ground truth locations (denoted GT), where the size of these boxes are fixed at $64 \times 64$. We can see that as the time step increases, the boxes start to converge to the parts of the birds. These success cases make sense because the training objective takes into
account the ground truth part locations. For the failure case, some of the bounding boxes do not converge to the ground truth parts. Nonetheless, some of the boxes still converge to the annotated parts. Although some bounding boxes do not fall on the object, LSTM has a robust mechanism of memory to memorize important parts. Thus classification does not critically depend on the detections at the final time step. This is also clear in the success case, where some boxes could have been refined into better positions and sizes.

Since no part annotations are provided for Cars-196, we cannot quantitatively compare the part predictions to ground truth. Instead, we visualize the average part locations that are predicted by our algorithm. Figure 3.7 shows the average image of the Cars-196 dataset and shows clusters of part locations predicted for the cars dataset. The centers represent the mean locations of parts and the circles represent the range of those centers. The part locations mostly align with the front of the car and the center of the image, where most of the car is present on average. The average image shows that most images of a car are taken from the front, which means we can expect some average part locations to align with the front of the car. Indeed, on average most parts align with the front of the car. All parts are near the center of the image, where most cars are present on average. Furthermore, the average part locations are relatively diverse, covering a majority of the average image rather than just a few locations. Finally, Figure 3.7 shows that our approach discovers visually diverse parts that are also discriminative, as desired for fine-grained categorization.
Figure 3.6: Sequence of Predicted Bounding Boxes for Several Images

(a) $t = 5$  
(b) $t = 10$  
(c) $t = 15$  
(d) GT  

(e) $t = 5$  
(f) $t = 10$  
(g) $t = 15$  
(h) GT  

(i) $t = 5$  
(j) $t = 10$  
(k) $t = 15$  
(l) GT

Figure 3.7: Average Image of Cars-196 and Map Showing Clusters of Parts

(a) Average Image  
(b) Parts Clusters
3.5 Summary

We presented a search-based framework with deep architectures for fine-grained recognition that achieves competitive results. We proposed a search-based architecture where the search space is defined on a convolutional feature map computed by a CNN, and the heuristic and successor functions are parameterized by a new deep network architecture called HSnet. HSnet is formulated with a built-in refinement mechanism to search for increasingly more informative parts and thus improve recognition, in addition to a robustness mechanism against wrongly-identified parts during inference. We specified two training settings, one where part location annotations are available and one where they are not available, where the latter is addressed with a determinantal point process loss for obtaining diverse proposals. Finally, our experimental results on Caltech-UCSD Birds 200-2011 and Cars-196 datasets demonstrated that sequential reasoning about object parts and removing background context are effective for fine-grained recognition.
Chapter 4: Adversarial LSTM Networks for Video Summarization

4.1 Introduction

In this chapter, we present an approach for video summarization, where the task of picking a subset of key frames from a limited number of videos is fine-grained recognition of summary key frames from videos. A wide range of applications require automated summarization of videos [141, 172], for instance, for saving time of human inspection, or enabling subsequent video analysis. Depending on the application, there are various distinct definitions of video summarization [117, 112, 113, 5, 156, 142, 40, 89, 75, 62, 110, 49]. In this chapter, we consider unsupervised video summarization and cast it as a key frame selection problem. Given a sequence of video frames, our goal is to select a sparse subset of frames such that a representation error between the video and its summary is minimal.

Our problem statement differs from other formulations considered in the literature, for example, when a particular domain of videos to be summarized is a priori known (e.g. first-person videos) [75], or when ground-truth annotations of key frames are provided in training data based on attention, aesthetics, quality, landmark presence, and certain object occurrences and motions [46].

Figure 4.1(a) provides an overview of our approach to selecting key frames from a given video. The key frame selector is learned so as to minimize a distance between features extracted from the video and the selected key frames. Following recent advances in deep learning [140, 163, 173], we extract deep features from both the video and selected sequence of key frames using a cascade of a Convolutional Neural Network (CNN)—specifically GoogleNet [143]—and Long Short-Term Memory Network (LSTM) [50, 140]. The CNN is grounded onto pixels and extracts deep features from a given frame. The LSTM then fuses a sequence of the CNN’s outputs for capturing long-range dependencies among the frames and produces its own deep feature representing the input sequence. Specifically, we use the (variational) auto-encoder LSTM [140, 70] as a suitable deep architecture for unsupervised learning of video features. Given a distance between the deep representations of the video and selected key frames, our goal is to optimize the frame selector such that this distance is minimized over training exam-
Recent work, however, demonstrates that specifying a suitable distance of deep features is difficult [88]. Hence, we resort to the generative adversarial framework [42], which extends the aforementioned video summarization network with an additional discriminator network. As shown in Figure 4.1(b), the decoder part of the summarizer is used to reconstruct a video from the sequence of selected key frames. Then, we use a discriminator, which is another LSTM, to distinguish between the original video and its reconstruction from the summarizer. The auto-encoder LSTM and the frame selector are jointly trained so as to maximally confuse the discriminator LSTM—in other words, they are cast in a role of the discriminator’s adversary—such that the discriminator has a high error rate in discriminating between the original and reconstructed videos. When this recognition error becomes maximum, we deem that the frame selector has learned to produce optimal video summarizations.

As we will show in this chapter, our approach allows for an effective regularization of generative-adversarial learning in terms of (i) limiting the total number of key frames that can be selected or (ii) maximizing visual diversity among the selected key frames. For a fair comparison
with related approaches to fully supervised video summarization (a different setting from ours that provides access to ground-truth key frame annotations in training), we also show how to effectively incorporate the available supervision as an additional type of regularization in learning.

Evaluation on four benchmark datasets, consisting of videos showing diverse events in first- and third-person views, demonstrates our competitive performance in comparison to fully supervised state-of-the-art approaches.

Our contributions include the following:

1. A new approach to unsupervised video summarization that combines variational autoencoders and generative-adversarial training of deep architectures.

2. The first specification of generative-adversarial training on high resolution video sequences.

In the following sections, Section 4.2 reviews prior work, Section 4.3.1 briefly introduces the generative adversarial network (GAN) and the variational autoencoder (VAE) models, Section 4.3.2 specifies main components of our approach, Section 4.3.3 formulates our end-to-end training, Section 4.4 describes variants of our approach differing in types of regularization we use in learning, and finally Section 4.5 presents our results.

4.2 Literature Review

This section reviews related problem formulations of video summarization, approaches to supervised and unsupervised video summarization, deep learning approaches, and work using the generative adversarial framework in learning.

Various Problem Formulations. Video summarization is a long-standing problem, with various formulations considered in the literature. For example, the video synopsis [113] tracks moving objects and then packs the identified video tubes into a smaller space-time volume. Montages [5, 156, 142] merge and overlap key frames into a single summary image. Both of these problem formulations, however, do not require that the video summary preserve the information about the temporal layout of motions in the video. Previous work has also studied hyperlapses where the camera viewpoint is being changed during the time-lapse for speeding-up or slowing-down certain parts of the input video [75, 62, 110, 49]. Our problem statement is closest to storyboards, representing a subset of representative video frames [40, 89]. However, except for [173, 163], existing approaches to generating storyboards do not take advantage of deep learning.
Supervised vs. Unsupervised Summarization. Supervised methods assume access to human annotations of key frames in training videos and seek to optimize their frame selectors so as to minimize loss with respect to this ground truth [41, 173, 172]. However, for a wide range of domains, it may be impossible to provide reliable and a sufficiently large amount of human annotations (e.g. military, nursing homes). These domains have been addressed with unsupervised methods, which typically use heuristic criteria for ranking and selecting key frames [92, 163, 68, 177, 139]. There have been attempts to use transfer learning for domains without supervision [173], but the surprisingly better performance of the transfer learning setting compared to the canonical setting, reported in Zhang et al. [173], suggests a high correlation of the domains for three training dataset and one test dataset, which is hard to ensure in real-world settings.

Deep Architectures for Video Summarization. In Zhang et al. [173], two LSTMs are used, one along the time sequence and the other in reverse from the video’s end, to select key video frames and trained by minimizing the cross-entropy loss on annotated ground-truth key frames with an additional objective based on determinantal point process (DPP) to ensure diversity of the selected frames. Our main differences are that we do not consider the key frame annotations and train our LSTMs using the unsupervised generative-adversarial learning. In Yang et al. [163], recurrent auto-encoders are learned to represent annotated temporal intervals in training videos, called highlights. In contrast, we do not require human annotations of highlights in training, and we do not perform temporal video segmentation (highlight vs non-highlight), but key frame selection.

Generative Adversarial Networks (GANs) have been used for image-understanding problems [42, 114, 128, 120] and frame prediction/generation [97, 150, 37]. However, we are not aware of their previous use for video summarization. In Larsen et al. [88], the discriminator output of a GAN is used to provide a learning signal for the variational auto-encoder (VAE). We extend this approach in three critical ways: (1) We specify a new variational auto-encoder LSTM, whereas their auto-encoder is not a recurrent neural network and thus cannot be used for videos; (2) Our generative-adversarial learning additionally takes into account the frame selector, a component not considered in Larsen et al. [88]; and (3) We formulate regularization of generative-adversarial learning that is suitable for video summarization.
4.3 Technical Approach

4.3.1 Review of VAE and GAN

**Variational Autoencoder** (VAE) [70] is a directed graphical model which defines a posterior distribution over the observed data, given an unobserved latent variable. Let \( e \sim p_e(e) \) be a prior over the unobserved latent variable and \( x \) be the observed data. One can interpret \( e \) as the encoding of \( x \) and define \( q(e|x) \) as the probability of observing \( e \) given \( x \). It is typical to set \( p_e(e) \) as the standard normal distribution. Similarly, \( p(x|e) \) identifies the conditional generative distribution for \( x \). Learning is done by minimizing the negative log-likelihood of the data distribution:

\[
-\log \frac{p(x|e)p(e)}{q(e|x)} = -\log (p(x|e)) + D_{KL}(q(e|x)\|p(e)).
\] (4.1)

For efficient learning, Kingma et al. [70] propose a reparameterization of the variational lower bound suitable for stochastic gradient descent.

**Generative Adversarial Network** (GAN) [42] is a neural network that consists of two competing subnetworks: (i) a ‘generator’ network (G) that generates data mimicking an unknown distribution and (ii) a ‘discriminator’ network (D) that discriminates between the generated samples and the true observations. The goal is to find a generator which fits the true data distribution while maximizing the probability of the discriminator making a mistake.

Let \( x \) be the true data sample, \( e \sim p_e(e) \) be the prior input noise and \( \hat{x} = G(e) \) be the generated sample. Learning is formulated as the following minimax optimization:

\[
\min_G \max_D \left[ \mathbb{E}_x[\log D(x)] + \mathbb{E}_e[\log (1 - D(\hat{x}))] \right],
\] (4.2)

where \( D \) is trained to maximize the probability of correct sample classification (true vs generated) and \( G \) is simultaneously trained to minimize \( \log (1 - D(\hat{x})) \).

4.3.2 Main Components of Our Approach

Our approach consists of training summarizer and discriminator recurrent networks, as illustrated in Figure 4.2. The selector LSTM (sLSTM) selects a subset of frames from the input sequence.
Figure 4.2: Main Components of Video Summarization Approach

$x$. The encoder LSTM (eLSTM) encodes the selected frames to a fixed-length feature vector $e$, which is then forwarded to the decoder LSTM (dLSTM) for reconstructing a video $\hat{x}$. The discriminator LSTM (cLSTM) classifies $\hat{x}$ as ‘original’ or ‘summary’ class. dLSTM and cLSTM form the generative adversarial network (GAN).

Given CNN’s deep features for every frame of the input video, $x = \{x_t : t = 1, \ldots, M\}$, the summarizer uses a selector LSTM (sLSTM) to select a subset of these frames, and then an encoder LSTM (eLSTM) to encode the sequence of selected frames to a deep feature vector, $e$. Specifically, for every frame $x_t$, sLSTM outputs normalized importance scores $s = \{s_t : s_t \in [0, 1], t = 1, \ldots, M\}$ for selecting the frame. The input sequence of frame features $x$ is weighted with these importance scores, and then forwarded to eLSTM. Note that in the special case of discretized scores, $s_t \in \{0, 1\}$, eLSTM receives only a subset of frames for which $s_t = 1$. The last component of the summarizer is a decoder LSTM (dLSTM), which takes $e$ as input, and reconstructs a sequence of features corresponding to the input video, $\hat{x} = \{\hat{x}_1, \hat{x}_2, \ldots, \hat{x}_M\}$.

The discriminator is aimed at distinguishing between $x$ and $\hat{x}$ as belonging to two distinct
classes: ‘original’ and ‘summary’. This classifier can be viewed as estimating a distance between \(x\) and \(\hat{x}\), and assigning distinct class labels to \(x\) and \(\hat{x}\) if their distance is sufficiently large. In this sense, the discriminator serves to estimate a representation error between the original video and our video summarization. While one way to implement the discriminator could be an energy-based encoder-decoder [178], in our experiments, a binary sequence classifier have shown better performance. Hence, we specify the discriminator as a classifier LSTM (cLSTM) with a binary-classification output.

Analogous to the generative adversarial networks presented in [42, 88], we have that dLSTM and cLSTM form the generative adversarial network (GAN). The summarizer and discriminator networks are trained adversarially until the discriminator is not able to discriminate between the reconstructed videos from summaries and the original videos.

### 4.3.3 Training of sLSTM, eLSTM, and dLSTM

This section specifies our learning of (i) Summarizer parameters, \(\{\theta_s, \theta_e, \theta_d\}\), characterizing sLSTM, eLSTM, and dLSTM; and (ii) GAN parameters, \(\{\theta_d, \theta_c\}\), defining dLSTM and cLSTM. Note that \(\theta_d\) are shared parameters between the summarizer and GAN.

As illustrated in Figure 4.3, our training is defined by four loss functions: 1) Loss of GAN, \(L_{\text{GAN}}\). 2) Reconstruction loss for the recurrent encoder-decoder, \(L_{\text{reconst}}\). 3) Prior loss, \(L_{\text{prior}}\), and 4) Regularization loss, \(L_{\text{sparsity}}\). The key idea behind our generative-adversarial training is to introduce an additional frame selector \(s_p\), governed by a prior distribution (e.g. uniform distribution), \(s_p \sim p(s_p)\). Sampling the input video frames with \(s_p\) gives a subset which is
passed to eLSTM, producing the encoded representation $e_p$. Given $e_p$, dLSTM reconstructs a
video sequence $\hat{x}_p$. We use $\hat{x}_p$ to regularize learning of the discriminator, such that cLSTM is
highly accurate on recognizing $\hat{x}_p$ as the ‘summary’ class, but that it confuses $\hat{x}$ as the ‘original’
class. Recall that the $\mathcal{L}_{\text{prior}}$ is imposed by the prior distribution over $e$ as in Equation (4.1).

Similar to the training of GAN models in [42, 88], we formulate an adversarial learning
algorithm that iteratively optimizes the following three objectives:

1. For learning \{\theta_s, \theta_e\}, minimize $(\mathcal{L}_{\text{reconst}} + \mathcal{L}_{\text{prior}} + \mathcal{L}_{\text{sparsity}})$.
2. For learning $\theta_d$, minimize $(\mathcal{L}_{\text{reconst}} + \mathcal{L}_{\text{GAN}})$.
3. For learning $\theta_c$, maximize $\mathcal{L}_{\text{GAN}}$.

In the following, we define $\mathcal{L}_{\text{reconst}}$ and $\mathcal{L}_{\text{GAN}}$, while the specification of $\mathcal{L}_{\text{sparsity}}$ is deferred
to Section 4.4.

**Reconstruction loss** $\mathcal{L}_{\text{reconst}}$: The standard practice in learning encoder-decoder networks is
to use the Euclidean distance between the input and decoded output, $\|x - \hat{x}\|_2$, for estimating the
reconstruction error. However, recent findings demonstrate shortcomings of this practice [88].
Hence instead, we define $\mathcal{L}_{\text{reconst}}$ based on the hidden representation in cLSTM—specifically,
the output of the last hidden layer of cLSTM, $\phi(x)$, for input $x$. Note that while $x$ is a sequence
of features, $\phi(x)$ represents a compact feature vector, capturing long-range dependencies in the
input sequence. Therefore, it seems more appropriate to use $\phi(x)$, rather than $x$, for specifying
$\mathcal{L}_{\text{reconst}}$.

Specifically, we formulate $\mathcal{L}_{\text{reconst}}$ as an expectation of a log-likelihood $\log p(\phi(x)|e)$, given
that $x$ has been passed through the frame selector $s$ and eLSTM, resulting in $e$:

$$
\mathcal{L}_{\text{reconst}} = \mathbb{E}[\log p(\phi(x)|e)],
$$

where expectation $\mathbb{E}$ is approximated as the empirical mean of training examples. In this chapter,
we consider $p(\phi(x)|e) \propto \exp(-\|\phi(x) - \phi(\hat{x})\|^2)$, while other non-Gaussian likelihoods are also
possible.

**Loss of GAN, $\mathcal{L}_{\text{GAN}}$:** Following [88], our goal is to train the discriminator such that cLSTM
classifies reconstructed feature sequences $\hat{x}$ as ‘summary’ and original feature sequences $x$ as
‘original’. In order to regularize this training, we additionally enforce that cLSTM learns to
classify randomly generated summaries $\hat{x}_p$ as ‘summary’, where $\hat{x}_p$ is reconstructed from a subset
of video frames randomly selected by sampling from a given prior distribution. In this chapter,
for this prior, we consider the uniform distribution. This gives the following loss:
Algorithm 3 Training SUM/GAN Model

1: **Input:** Training video sequences
2: **Output:** Learned parameters \( \{ \theta_s, \theta_e, \theta_d, \theta_c \} \).
3: Initialize all parameters \( \{ \theta_s, \theta_e, \theta_d, \theta_c \} \).
4: for max number of iterations do
5: \( X \leftarrow \) mini-batch from CNN feature sequences
6: \( S \leftarrow \) sLSTM(\( X \)) // select frames
7: \( E = \) eLSTM(\( X, S \)) // encoding
8: \( \hat{X} = \) dLSTM(\( E \)) // reconstruction
9: \( S_p \leftarrow \) draw samples form the uniform distribution
10: \( E_p = \) eLSTM(\( X, S_p \)) // encoding
11: \( X_p = \) dLSTM(\( E_p \)) // reconstruction
12: // Updates using Stochastic Gradient:
13: \( \{ \theta_s, \theta_e \} \) \( \leftarrow \) \( -\nabla (L_{\text{reconst}} + L_{\text{prior}} + L_{\text{sparsity}}) \)
14: \( \{ \theta_d \} \) \( \leftarrow \) \( -\nabla (L_{\text{reconst}} + L_{\text{GAN}}) \)
15: \( \{ \theta_c \} \) \( \leftarrow \) \( +\nabla (L_{\text{GAN}}) \) // maximization update
16: end for

\[
L_{\text{GAN}} = \log(\text{cLSTM}(x)) + \log(1 - \text{cLSTM}(\hat{x})) + \log(1 - \text{cLSTM}(\tilde{x})),
\]

(4.4)

where \( \text{cLSTM}(\cdot) \) denotes the binary softmax output of \( \text{cLSTM} \).

Given the above definitions of \( L_{\text{reconst}} \) and \( L_{\text{GAN}} \), as well as \( L_{\text{sparsity}} \) explained in Section 4.4, we update the parameters \( \theta_s, \theta_e, \theta_d \) and \( \theta_c \) using the Stochastic Gradient Variational Bayes estimation [71, 70], adapted for recurrent networks [24]. Algorithm 3 summarizes all steps of our training. Note that Algorithm 3 uses capital letters to denote a mini-batch of the corresponding variables with small-letter notation in the previous text.

4.4 Variants of our Approach

This section explains our regularization of learning. We use the following three types of regularization, which define the corresponding variants of our approach.

**Summary-Length Regularization** penalizes having a large number of key frames selected in the summary as
where $M$ is the total number of video frames, and $\sigma$ is an input hyper-parameter representing a percentage of frames that we expect to be selected in the summary. When our approach uses $\mathcal{L}_{\text{sparsity}}$, we call it SUM-GAN.

**Diversity Regularization** enforces selection of frames with high visual diversity, in order to mitigate redundancy in the summary. In this chapter, we use two standard definitions for diversity regularization, namely, (i) Determinantal Point Process (DPP) [143, 41, 173]; and (ii) Repelling regularizer (REP) [178].

Following Zhang et al. [173], our DPP based regularization is defined as

$$
\mathcal{L}_{\text{sparsity}}^{\text{dpp}} = -\log(P(s))
$$

(4.6)

where $P(s)$ is a probability that DPP assigns to the selection indicator $s$. We compute $P(s; L) = \frac{\det(L(s))}{\det(L-I)}$, where $L$ is an $M \times M$ similarity matrix between every two hidden states in eLSTM, $I$ is an identity matrix and $L(s)$ is a smaller square matrix, cut down from $L$ given $s$. Let $e_t$ be the hidden state of eLSTM at time $t$. For time steps $t$ and $t'$ the pairwise similarity values are defined as $L_{t,t'} = s_t s_{t'} e_t e_{t'}$. When our approach uses $\mathcal{L}_{\text{sparsity}}^{\text{dpp}}$, we call it SUM-GAN$_{\text{dpp}}$.

For repelling regularization, we define

$$
\mathcal{L}_{\text{sparsity}}^{\text{rep}} = \frac{1}{M(M-1)} \sum_t \sum_{t' \neq t} \left( \frac{e_t^\top e_{t'}}{\|e_t\|\|e_{t'}\|} \right)
$$

(4.7)

and call this variant of our approach as SUM-GAN$_{\text{rep}}$.

**Keyframe Regularization** is specified for the supervised setting where ground-truth annotations of key frames are provided in training. This regularization enables a fair comparison of our approach with recently proposed supervised methods. Note that we here consider importance scores as 2D softmax outputs $\{s_t\}$, rather than scalar values as introduced in Section 4.3.2. We define the sparsity loss as the cross-entropy loss:

$$
\mathcal{L}_{\text{sparsity}}^{\text{sup}} = \frac{1}{M} \sum_t \text{cross-entropy}(s_t, \hat{s}_t)
$$

(4.8)
We call this variant of our approach as SUM-GAN\textsubscript{sup}.

4.5 Experiments

4.5.1 Setup

**Datasets.** We evaluate our approach on four datasets: SumMe [46], TVSum [139], Open Video Project (OVP) [106, 19], and Youtube [19]. First, SumMe consists of 25 user videos. The videos capture multiple events such as cooking and sports. The video contents are diverse and include both first-person and third-person camera views. The video lengths vary from 1.5 to 6.5 minutes. The dataset provides frame-level importance scores. Next, TVSum contains 50 videos from YouTube. The videos are selected from 10 categories in the TREC Vid Multimedia Event Detection (MED) (5 videos per category). The video lengths vary from 1 to 5 minutes. Similar to SumMe, the video contents are diverse and include both ego-centric and third-person camera views. For OVP, we evaluate on the same 50 videos used in [19]. The videos are from various genres (e.g. documentary, educational), and their lengths vary from 1 to 4 minutes. Finally, the YouTube dataset includes 50 videos collected from websites. The duration of the videos range from 1 to 10 minutes, and the contents include cartoons, news and sports.

**Evaluation Setup.** For a fair comparison with the state of the art, the keyshot-based metric proposed in Zhang et al. [173] is used for evaluation. Let $A$ be the generated keyshots and $B$ the user-annotated keyshots. The precision and recall are defined based on the amount of temporal overlap between $A$ and $B$ as follows:

$$\text{precision} = \frac{\text{duration of overlap between } A \text{ and } B}{\text{duration of } A}$$

$$\text{recall} = \frac{\text{duration of overlap between } A \text{ and } B}{\text{duration of } B}$$

Finally, the harmonic mean F-score is used as the evaluation metric. We follow the steps in Zhang et al. [173] to convert frame-level scores to key frames and key shot summaries and vice versa in all datasets. To generate key shots for datasets that only provide key frame scores, the videos are initially temporally segmented into disjoint intervals using KTS [111]. The resulting intervals are ranked based on their importance score where the importance score of an interval is...
equal to the average score of the frames in that interval. A subset of intervals are selected from the ranked intervals as keyshots such that the total duration of the generated key shots are less than 15% of the duration of the original video.

For datasets with multiple human annotations (in the form of key shots or key frames), we follow the standard approach described in [47, 139, 173] to create a single ground-truth set for evaluation. While evaluating our SUM-GAN\textsubscript{sup} model, we used the same train, validation and test splits as in Zhang et al. [173]. For fair comparison, we also run it for five different random splits and report the average performance.

**Implementation Details:** For fair comparison with [173], we choose to use the output of pool5 layer of the GoogLeNet network [143] (1024-dimensions), trained on ImageNet [127], for the feature descriptor of each video frame. We use a two-layer LSTM with 1024 hidden units at each layer for the discriminator LSTM (cLSTM). We use two two-layer LSTMs with 2048 hidden units at each layer for the eLSTM and dLSTM respectively. It is shown in Srivastava et al. [140] that a decoder LSTM that attempts to reconstruct the reverse sequence is easier to train. Consequently, our dLSTM reconstructs the feature sequence in reverse order. Note that while presenting $x$ and $\hat{x}$ as the cLSTM input, both sequences should have similar ordering in time.

We initialize the parameters of the eLSTM and dLSTM with the parameters of a pre-trained recurrent autoencoder model trained on feature sequences from original videos. We discovered that this helps to improve the overall accuracy and also results in faster convergence.

The sLSTM network is a two-layer bidirectional LSTM with 1024 hidden units. The output is a 2-dimensional softmax layer in the case of SUM-GAN\textsubscript{sup}. We train our framework with the Adam optimizer using the default parameters.

### 4.5.2 Baselines

It is important to point out that considering the generative structure of our approach and the definition of the update rules in Alg. 3, it is not possible to entirely replace subnetworks of our model baselines. Instead, in addition to different variations of our approach defined in Section 4.4, we also evaluate the following baselines:

1. SUM-GAN\textsubscript{bas}: does not use the sparsity regularization
2. SUM-GAN\textsubscript{w/o-GAN}: does not include $\mathcal{L}_{GAN}$ while updating $\{\theta_d\}$
3. SUM-GAN\textsubscript{w/o-s$_p$}: replaces Equation (4.4) with $\mathcal{L}_{GAN} = \log(cLSTM(x)) + \log(1 - cLSTM(\hat{x}))$, in other words, does not consider random summaries while training
Table 4.1: Comparison of Different Variations on Benchmark Datasets

<table>
<thead>
<tr>
<th>Method</th>
<th>SumMe</th>
<th>TVSum</th>
<th>OpenVideo</th>
<th>YouTube</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUM-GAN</td>
<td>38.7</td>
<td>50.8</td>
<td>71.5</td>
<td>58.9</td>
</tr>
<tr>
<td>SUM-GAN$_{bas}$</td>
<td>35.7</td>
<td>50.1</td>
<td>69.8</td>
<td>57.1</td>
</tr>
<tr>
<td>SUM-GAN$_{w/o-GAN}$</td>
<td>34.6</td>
<td>49.5</td>
<td>69.3</td>
<td>56.9</td>
</tr>
<tr>
<td>SUM-GAN$_{w/o-s_p}$</td>
<td>37.2</td>
<td>50.4</td>
<td>71.5</td>
<td>58.4</td>
</tr>
<tr>
<td>SUM-GAN$_{dpp}$</td>
<td>38.5</td>
<td>51.9</td>
<td>72.3</td>
<td>59.6</td>
</tr>
<tr>
<td>SUM-GAN$_{sup}$</td>
<td>39.1</td>
<td>51.7</td>
<td>72.8</td>
<td>60.1</td>
</tr>
<tr>
<td>SUM-GAN$_{sup}$</td>
<td>41.7</td>
<td>56.3</td>
<td>77.3</td>
<td>62.5</td>
</tr>
</tbody>
</table>

### 4.5.3 Quantitative Results

Table 4.1 summarizes the accuracy of different variations of our approach on benchmark datasets. The result for SUM-GAN is reported for $\sigma = 0.3$. As is expected, the model with additional frame-level supervision, SUM-GAN$_{sup}$, outperforms the unsupervised variants by 2-5%.

One interesting observation is that although explicit regularization of the model with ‘diversity regularizers’ (SUM-GAN$_{dpp}$ and SUM-GAN$_{rep}$) performs slightly better than the variant of our model with ‘length regularizer’ (SUM-GAN) the difference is not statistically significant. Furthermore, in the case of SumMe, SUM-GAN performs better than SUM-GAN$_{rep}$. This is particularly important because it verifies our main hypothesis that a good summary should include a subset of frames which provide similar content representation as of the original frame sequences. This suggests that if we constrain the summary to be shorter in length, implicitly the frames will be diverse. We also observe that SUM-GAN$_{dpp}$ performs better than SUM-GAN$_{rep}$ in all four datasets. We believe that this is mainly because of the fact that unlike the repelling regularizer, DPP is non-linear and can reinforce stronger regularization. Comparing the accuracy of SUM-GAN$_{w/o-GAN}$ with SUM-GAN shows that training with the combined losses from the VAE and GAN improves the accuracy.

We are particularly interested in comparing our performance in contrast with prior unsupervised and supervised methods. This comparison is presented in Table 4.2 for unsupervised ap-
Table 4.2: Comparison of State-of-the-Art Unsupervised Approaches

<table>
<thead>
<tr>
<th>Method</th>
<th>SumMe</th>
<th>TVSum</th>
<th>OpenVideo</th>
<th>YouTube</th>
</tr>
</thead>
<tbody>
<tr>
<td>De Avila 2011 [19]</td>
<td>33.7</td>
<td>-</td>
<td>70.3</td>
<td>59.9</td>
</tr>
<tr>
<td>Li 2010 [92]</td>
<td>26.6</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Khosla 2013 [68]</td>
<td>-</td>
<td>36.0</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Song 2015 [139]</td>
<td>26.6</td>
<td>50.0</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Furini 2010 [34]</td>
<td>-</td>
<td>-</td>
<td>63.4</td>
<td>-</td>
</tr>
<tr>
<td>Mundur 2006 [104]</td>
<td>-</td>
<td>-</td>
<td>57.6</td>
<td>-</td>
</tr>
<tr>
<td>Zhao 2014 [177]</td>
<td>-</td>
<td>46.0</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>SUM-GAN&lt;sub&gt;dpp&lt;/sub&gt;</td>
<td><strong>39.1</strong></td>
<td><strong>51.7</strong></td>
<td><strong>72.8</strong></td>
<td><strong>60.1</strong></td>
</tr>
</tbody>
</table>

Table 4.3: Comparison of State-of-the-Art Supervised Approaches

<table>
<thead>
<tr>
<th>Method</th>
<th>SumMe</th>
<th>TVSum</th>
<th>OpenVideo</th>
<th>YouTube</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gygli 2015 [47]</td>
<td>39.7</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Zhang 2016 [172]</td>
<td>40.9</td>
<td>-</td>
<td>76.6</td>
<td>60.2</td>
</tr>
<tr>
<td>Gygli 2014 [46]</td>
<td>39.3</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Zhang 2016 [173]</td>
<td>38.6</td>
<td>54.7</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Gong 2014 [41]</td>
<td>-</td>
<td>-</td>
<td><strong>77.7</strong></td>
<td>60.8</td>
</tr>
<tr>
<td>SUM-GAN&lt;sub&gt;sup&lt;/sub&gt;</td>
<td><strong>41.7</strong></td>
<td><strong>56.3</strong></td>
<td><strong>77.3</strong></td>
<td><strong>62.5</strong></td>
</tr>
</tbody>
</table>

Table 4.4: Comparison of State-of-the-Art Approaches With Training Data Augmentation

<table>
<thead>
<tr>
<th>Method</th>
<th>SumMe</th>
<th>TVSum</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zhang 2016 [172]</td>
<td>40.9</td>
<td>-</td>
</tr>
<tr>
<td>Zhang 2016 [173]</td>
<td>42.9</td>
<td>59.6</td>
</tr>
<tr>
<td>SUM-GAN</td>
<td>41.7</td>
<td>58.9</td>
</tr>
<tr>
<td>SUM-GAN&lt;sub&gt;rep&lt;/sub&gt;</td>
<td>42.5</td>
<td>59.3</td>
</tr>
<tr>
<td>SUM-GAN&lt;sub&gt;dpp&lt;/sub&gt;</td>
<td>43.4</td>
<td>59.5</td>
</tr>
<tr>
<td>SUM-GAN&lt;sub&gt;sup&lt;/sub&gt;</td>
<td>43.6</td>
<td>61.2</td>
</tr>
</tbody>
</table>

...approaches and Table 4.3 for supervised approaches, where the reported state-of-the-art results are obtained from their published results. (Note that in Table 4.3, results from [172, 41] use only 39...
As shown, our unsupervised SUM-GAN model outperforms all unsupervised approaches in all datasets. For SumMe, our approach is almost 5% better than the state-of-the-art unsupervised approaches. More importantly, the accuracy of SUM-GAN is comparably close to the supervised methods in TVSum, OVP, and YouTube datasets.

Comparing with the state-of-the-art supervised approaches, our supervised variant, SUM-GAN, outperforms in all datasets except OVP. Even in the case of OVP, we are statistically close to the best reported accuracy with 0.4% margin. We hypothesize that the accuracy boost is mainly because of the additional learning signal from the cLSTM. Note that the discriminator observes a longer sequence and classifies based on a learned semantic representation of the feature sequence. This enables the discriminator to provide a more informative signal regarding the importance of the frames for content similarity.

Zhang et al. [173] augment the SumMe and TVSum datasets with OVP and YouTube datasets and improve the accuracy on SumMe and TVSum. Table 4.4 shows the accuracy results in comparison with results reported in [173] when training dataset is augmented. (Note that for Zhang et al. [173], results without domain adaptation are reported.) Except for SUM-GAN, which we use 80% of the target dataset in training, for the unsupervised variants of our approach we use all four datasets in training. The most important observation is that one of our unsupervised variations, SUM-GAN, outperforms the state of the art in SumMe. This shows that if trained with more unsupervised video data, our model is able to learn summaries which are competitive with the models trained using key frame annotations.

Finally, we evaluate the performance of our approach for different percentages of $\sigma$ values for our SUM-GAN model. Figure 4.4 shows the resulting F-score values for different $\sigma$’s on four different datasets. While the performance is consistent for $0.3 \leq \sigma \leq 0.5$, it drops rapidly as $\sigma \rightarrow 1$ or $\sigma \rightarrow 0$.

### 4.5.3.1 Comparison with Shallow Features

We verified the generalizability of our video summarization approach to non-deep features by evaluating our model with the shallow features employed in Zhang et al. [172, 173] Table 4.5 shows the performance of our model compared to the state-of-the-art models that use shallow features. Besides the reported results in Zhang et al. [172] for TvSum, where the shallow features outperform the deep features, our model consistently performs better than the state of the art. Unlike Zhang et al. [172], our model grounded on deep features still performs better than the
same model grounded on shallow features.

4.5.4 Qualitative Results

To better illustrate the temporal selection pattern of different variations of our approach, we demonstrate the selected frames on an example video in Figure 4.5. The blue background shows the frame-level importance scores. The colored regions are the selected subsets for different methods. The visualized key frames for different variants support the result presented in Ta-
Figure 4.5: Example Summaries from a Sample Video in TvSum
ble 4.1. Despite small variations, all four approaches cover the temporal regions with high frame-level score. Most of the failure cases occurred in videos which consist of frames with very slow motions and no scene-change.

4.6 Summary

We proposed a generative architecture based on variational recurrent auto-encoders and generative adversarial networks for unsupervised video summarization to select a subset of key frames. Our main hypothesis is that the learned representation of the summary video and the original video should be similar. The summarizer aims to summarize the video such that the discriminator is fooled and the discriminator aims to recognize the summary videos from original videos. The entire model is trained in an adversarial manner where the GAN’s discriminator is used to learn a discrete similarity measure for training the recurrent encoder/decoder and the frame selector LSTMs. Variations of our approach are defined using different regularizations. Evaluation on benchmark datasets demonstrate that all unsupervised variations of our approach outperform the state of the art in video summarization by 2-5% and provides a comparable accuracy to the state-of-the-art supervised approaches. We also verified that the supervised variation of our approach outperforms the state of the art by 1-4%. 
Chapter 5: Part-Based Generative Adversarial Networks for Human 3D Pose Estimation

5.1 Introduction

In this chapter, we present an approach for human 3D pose estimation where predicting specific joint locations is fine-grained recognition of joints as parts from images. Predicting 3D locations of human-body joints from RGB images and videos (a.k.a. 3D pose estimation) is a longstanding problem in computer vision [52, 107, 32, 101], with a variety of applications including surveillance, human-computer interaction, motion capture and virtual reality. This is a very challenging problem, due to, in part, large variations of in body-part layouts, and self-occlusions of body parts in 2D images. To address these challenges, some previous methods resort to additional data beyond 2D images, such as multiple viewpoints [4, 51] and depth data [116, 169, 134]. This chapter focuses on recovering human 3D poses from image and video data only.

With the recent advent of large datasets with detailed annotations of human poses [56], there has been an increased interest in developing deep architectures for 3D pose estimation. These approaches can be generally divided into two frameworks: (i) discriminative [2, 12, 64, 123, 148], where the pose is directly regressed from image data; and (ii) generative [35, 135, 149], where the pose space of plausible skeleton configurations is searched to align with image data. The former typically use multiple streams of processing in their deep architectures, resulting in arguably very complex models that may not be easy to train. The latter usually search for the best matching 3D pose from a library of exemplars to the pose extracted from the image, and hence have difficulties generalizing to new 3D poses beyond the exemplars. The most recent approaches [180, 158] also demonstrate performance improvements when the dataset is augmented with auxiliary data and tasks (e.g., mixing in 2D pose estimation). However, we solely focus on human 3D pose estimation in this chapter.

In this chapter, we approach human 3D pose estimation from the generative framework. We specify a novel Generative Adversarial Network (GAN) for human 3D pose estimation. A GAN consists of two competing subnetworks: i) a ‘generator’ network which generates data from an
unknown distribution, and ii) a ‘discriminator’ network which discriminates between the generated samples and the true observations. The goal of learning a GAN is to estimate a generator that models the true data distribution while maximizing the probability of the discriminator making a mistake. GANs have been recently demonstrated as state-of-the-art performers for a wide range of computer vision problems [42, 114, 128, 120, 97, 150, 37]. However, to the best of our knowledge, GANs have not been used for 3D pose estimation.

Our key contribution is leveraging GANs to replace manually constructed 3D pose libraries used by many generative approaches to 3D pose estimation. To this end, we introduce a novel generator network within the GAN aimed at accounting for spatiotemporal dependences between body joints in a sequence of video frames. Our generator represents a deep architecture of spatial and temporal Long-Short Term Memory (LSTM) networks [50, 140], where each LSTM predicts a 3D location of the corresponding joint conditioned on predictions from other LSTMs. After modeling human 3D poses with the GAN, we combine the GAN with Variational Autoencoder (VAE) [70] to learn a projection of images to a space of human 3D poses learned by the GAN. A VAE is a directed graphical model that defines a posterior distribution over observed data, given an unobserved latent variable.

Figure 5.1 illustrates a high level overview of our approach. We learn a generative model of human 3D poses via a novel Generative Adversarial Network (GAN), whose generator represents a deep architecture of LSTMs taking into account spatiotemporal dependences of 3D locations of body joints. We use a variational autoencoder (VAE) to project the input image to the space of 3D poses learned by our generative model.

GANs are suitable for our purposes for a number of reasons. First, they have been shown to
learn a smooth latent representation from training data [42]. This motivates our hypothesis that such a smooth representation helps with generalization to new 3D poses never seen in training data. Second, their learning formulation implicitly alleviates overfitting, even when training data may not uniformly represent all possible 3D poses.

Our contributions include the following:

1. Modeling human 3D poses using GANs.
2. Generating human 3D poses with a novel generator for GAN consisting of spatial and temporal LSTMs.
3. 3D pose estimation by combining a VAE for encoding images with GANs that generate 3D poses.

In the following sections, Section 5.2 places our approach in the context of prior work, Section 5.3.1 reviews GANs and VAE, Section 5.3.2 specifies our novel generator architecture, and Section 5.4 presents our results.

5.2 Literature Review

Deep Regression 3D Pose Estimation. A majority of previous deep learning approaches formulate the problem of 3D pose estimation as regression from 2D image to 3D pose. Li et al. [91] train a multi-task regression convolutional neural network to predict 3D pose directly from images. Tekin et al. [146] work with spatiotemporal features of an image sequence to learn a regression model. In contrast to these and other regression approaches, we take a generative approach: we first train a GAN to model 3D poses and then train a VAE to map a 2D image to a 3D pose.

Exemplar-Based 3D Pose Estimation. Exemplar-based methods have also been explored in previous approaches [133]. One of the main challenges of exemplar-based approaches is generalization to novel poses outside of the training set. Jiang et al. [61] address generalization by matching upper and lower bodies individually. Yasin et al. [168] use an energy minimization approach to adapt exemplars to better match image measurements. Rogez et al. [122] synthesize new 2D images through image-based rendering. Other approaches warp from 2D image descriptors to 3D exemplars, such as shape context [2, 102] or silhouette features [55]. In our work, instead of learning a dictionary of exemplars, we train a generative model for sampling human 3D poses via GANs. The GAN training has been shown to yield a smooth underlying
distribution of data [42], which is likely to help us generalize well to never-seen human poses.

**Generative Adversarial Networks.** GANs have been recently explored for image modeling and understanding [42, 114, 128, 120]. More recently, GANs have been also applied to generating images from semantic segmentation data [57]. GANs have also been combined with variational autoencoders [88] for recognizing 3D shapes [157]. Perhaps most similar to our work is using adversarial training to learn constraints for human 2D poses [18]. However, we are not aware of any prior work that uses GANs for generative modeling of human 3D poses.

**Part-Based Modeling.** Part-based models and graphical modeling have been extensively studied in the past few decades [29]. While part-based models were the state of the art in object detection before deep learning [27, 28], they have been also recently combined with deep learning [39, 59]. However, we are not aware of generative LSTM graphical models that generate human 3D poses in addition to the tasks of prediction and understanding.

5.3 Technical Approach

5.3.1 Models

In Section 4.3.1, we provided a review of Generative Adversarial Networks (GAN) and Variational Autoencoders (VAE). In this section, we briefly review GAN and VAE again, and then specify how we change and merge them to yield our technical approach.

Figure 5.2 illustrates an overview of our architecture. We first train a GAN on human 3D poses for the ability to generate novel poses \( \hat{x} \). We then train VAE to project input images \( y \) onto the latent space of human 3D poses \( z \). During test time, an input image \( y \) is mapped to a 3D-pose latent representation \( z = E(y) \). The 3D pose is then recovered through the generator \( \hat{x} = G(z) \). Our novelty lies in the generator \( G \) (see Section 5.3.2).

5.3.1.1 Generative Adversarial Network

A GAN is a neural network that consists of two competing subnetworks: (i) a generator network \( G \) which generates data from an unknown distribution (e.g. ground-truth 3D poses) and (ii) a discriminator network \( D \) which discriminates between the generated samples from \( G \) and the ones from the unknown true distribution. The goal of learning a GAN is to find a generator \( G \) which models the true data distribution while maximizing the probability of the discriminator.
Figure 5.2: Overall GAN-VAE Architecture for Human 3D Pose Estimation

Let $x$ be the true data sample (in our case, a 3D pose), $z \sim p_z(z)$ be a randomly sampled noise vector from a prior noise distribution $p_z(z)$, and $\hat{x} = G(z)$ be the generated sample. Learning a GAN is specified as minimizing the following loss function:

$$L_{GAN} = \log D(x) + \log(1 - D(\hat{x})).$$  \hspace{1cm} (5.1)

$D$ is trained to maximize the probability of correct sample classification (true/real vs. generated/fake), and simultaneously $G$ is trained to minimize $\log(1 - D(\hat{x}))$.

### 5.3.1.2 GAN Details

The ground-truth 3D pose, $x$, is parameterized as a vector of $3K$ elements, where $K$ is the number of joints in a human skeleton. The elements of $x$ are normalized in the range of $[-1, 1]$. We interpret each 3D point as an offset from a reference 3D point, where the spherical coordinate system is used to parameterize the offsets $(r, \theta, \phi)$. For instance, the left elbow is parameterized as an offset to the left shoulder. These offsets form a tree, whose root represents the hip. The root has no offset.

The noise vector sampled from the latent space, $z$, is a 100-dimensional vector, sampled from a multivariate Gaussian distribution centered at 0, and characterized by a diagonal covariance with standard deviation 1.

We defer the specification of our generator $G$ to Section 5.3.2. Our discriminator $D$ mirrors $G$ with a final output from a sigmoid layer to yield a probability estimate of whether the input is making a mistake.
“real” or “fake”.

5.3.1.3 Variational Autoencoder

Our GAN is able to generate 3D points from noise, but to go from the image to 3D points for 3D pose estimation, we need a mapping from images to 3D points. We use a VAE to map an input image $y$ to an encoded representation $z$, which is governed by the underlying distribution $p_z(z)$. The encoded vector $z$ is then used to generate 3D points $\hat{x}$. This approach is similar to Wu et al. [157].

Our VAE consists of encoder $E$ and generator $G$ networks aimed at modeling a posterior distribution $p(y|z)$ of observed data, $y$, given an unobserved latent variable, $z$, where $z = E(y)$ and $\hat{x} = G(E(y))$. Let $z \sim p_z(z)$ be the unobserved latent variable sampled from distribution $p_z(z)$. The learning goal for the VAE is to minimize a loss function $L_{VAE}$, defined in terms of the KL divergence between $q(z|y)$ and $p_z(z)$ and the reconstruction error between the reconstructed output and the ground-truth $\|G(E(y)) - x\|_2$:

$$L_{VAE} = \|G(E(y)) - x\|_2 + D_{KL}(q(z|y)||p(z)).$$  \hspace{1cm} (5.2)

The second term in (5.2) is a regularization term aimed at reducing the difference between the variational distribution $q(z|y)$ and prior $p_z(z)$ so that the generator $G$ can sample the latent representation $z$ from a distribution as similar as possible to $p_z(z)$.

The astute reader will note that our formulation is technically not an autoencoder since $G(E(y)) = \hat{x} \neq \hat{y}$ where $\hat{y}$ would be a reconstructed image from $y$. While this may be the case, the literature still refers to this as a VAE in some cases [157], thus we will put this technicality away as a side note.

5.3.1.4 VAE Details

Our encoder $E$ is a series of convolutions that map an input sequence of images to the latent space governed by $p_z(z)$. Since we work with video clips, rather than independent single images, we use a series of 3D convolutions along space and time in $E$. The encoder architecture is similar to Wu et al. [157], except our input image dimensionality is $256 \times 256$ and our third dimension is time. We operate on a short video segment at a time. The resulting $z$ is then passed to
the generator $G$ to produce the 3D pose estimate $\hat{x}$. The estimated 3D pose should match the ground-truth 3D pose of the first frame of the input video clip.

It is worth noting that, in addition to the above specification of $E$, we also conducted experiments with the two-stream networks [26] (appearance from video and motion from optical flow). However, we were not able to get competitive performance with the two-stream networks, and hence do not discuss this variant of $E$ in the paper.

5.3.1.5 Our Architecture

We combine GAN and VAE into GAN-VAE and train it using the following loss function that incorporates the above losses of the GAN and the VAE networks:

$$L = L_{GAN} + L_{VAE}$$ (5.3)

Technically, the GAN is first trained to produce a generative model of 3D poses. The VAE is then trained to encode images to 3D poses.

5.3.2 GAN Generator: An Ablation Study

We specify a novel generator $G$ for the GAN, inspired by traditional part-based graphical modeling of human poses. Our novelty is in specifying $G$ as a deep architecture of LSTMs, each aimed at modeling spatiotemporal dependences between body joints for reliable 3D pose estimation. While there is a host of literature on part-based graphical modeling of human poses, it is not clear how to integrate them with our GAN-VAE. It is also unclear whether the best-performing graphical models of prior work would remain best performers once integrated with the GAN-VAE. Therefore, in this paper, we study a range of different architectures for $G$, as illustrated in Figure 5.3. (Skeletons are simplified for clarity.) These variants of $G$, denoted as V1–V6, will allow us to conduct ablation experiments and provide empirical insights on how our performance improves as $G$ becomes more complex. Below, we specify V1–V6, where $K = 32$ denotes the number of joints on the human skeleton.

The following is a description of Figure 5.3: (a) V1: Mutilayer perceptron predicts all joints simultaneously and independently of time. (b) V2: Temporal LSTMs (tLSTMs shown as the boxes $t$) predict the corresponding joints (one tLSTM per joint) simultaneously and independently at a given video frame $t$. Each tLSTM uses its memory from a previous frame $t - 1$
for the prediction at \( t \). (c) V3: At \( t \), all joints are predicted sequentially using a spatial LSTM (sLSTM shown as the box \( s \)), where the sLSTM uses its memory from a previous prediction to make the subsequent prediction. Note that the sLSTM is connected to a tLSTM aimed at capturing temporal dependences on predictions made at \( t - 1 \). (d) V4: At \( t \), each joint is predicted by the corresponding tLSTM, where all of the tLSTMs are connected in a tree for capturing spatial dependencies between the joints. (e) V5 has fewer LSTMs than V4. At \( t \), joints of the human limbs and other body parts are predicted sequentially using the corresponding sLSTMs, which in turn are connected to a tLSTM for temporal smoothness with predictions made at \( t - 1 \). (f) V6 is similar to V5 except the two pairs of sLSTMs for sequentially predicting joints of symmetrical limbs are connected.

**V1: Multilayer Perceptron.** In this variant, \( G \) is a multilayer perceptron. \( z \) is input to the multilayer perceptron for estimation of joint locations. There are no recurrent connections in V1, and hence temporal smoothness across consecutive video frames is not taken into account. V1 represents a very simple baseline to test the contribution of LSTMs in the following architectures of \( G \).

**V2: Independent LSTM Predictions.** In this variant, \( z \) is input into \( K \) independent tem-
poral LSTMs (tLSTMs), where each tLSTM is responsible for predicting a 3D location of the corresponding joint over time. Since tLSTMs are recurrent neural networks, they account for temporal long- and short-range temporal dependencies of consecutive joint locations. While in V2 each prediction of a joint location is independent of another, note that each joint location is conditioned on its past location, which is likely to improve temporal smoothness of the 3D pose relative to V1. V2 tests the value of using recurrent neural networks versus the multilayer perceptron for 3D pose estimation.

V3: Sequential LSTM Predictions. In this variant, there are two LSTMs: a temporal LSTM (tLSTM) and a spatial LSTM (sLSTM). The reason for having two LSTMs in V2 is that one handles time and the other handles space through their respective recurrent connections. The input $z$ is first passed to the tLSTM, where recurrent connections correspond to frames in the video. The output of the tLSTM for a particular frame is passed to the sLSTM, where recurrent connections correspond to the $K$ joints. The sLSTM is run sequentially for $K$ steps, each estimating the location of a particular joint, conditioned on the previous joint predictions. The sequence order of predicting the $K$ joints is pre-defined. V3 evaluates the merit of using an LSTM’s recurrent memory for making sequential predictions of joint locations at a given time $t$, rather than using multiple independent LSTMs per joint as in V2.

V4: Graph-Based LSTM Predictions. This variant is similar to V2, but instead of having independent tLSTMs per joint, the tLSTMs are also connected in a tree configuration. The tree is specified by a human skeleton model, where we choose the hip as the root, and the spine and limbs branch out of the root. For instance, the tLSTM for the left arm is conditioned on the output of the tLSTM for the left shoulder. Like V2, the recurrent connections for each tLSTM correspond to the time sequence in the video. Variant V4 evaluates the benefit of predicting joints conditioned on previous predictions using a tree-structured model, rather than independent predictions as in V2. Another purpose of V4 is to compare performance of the tree-structured configuration of tLSTMs with the sequential predictions of V3.

V5: Limbs-Sequential LSTM Predictions. This variant combines V3 and V4. The input $z$ is first fed into a tLSTM, where recurrent connections correspond to time steps in the video as in V3. The output of the tLSTM is then passed to an sLSTM responsible for predicting the joints along the spine. The output of that sLSTM is fed into four independent sLSTMs, each sequentially predicting joints for the corresponding limbs: left arm, right arm, left leg and right leg. Each sLSTM predicts the joints in a pre-defined order. Variant V5 shows the merit of combining one tLSTM for time with several sLSTMs connected in a certain configurations,
each sequentially predicting joints of the respective body part.

**V6: Dependent Limbs-Sequential LSTM Predictions.** Our final variant is similar to V5 except that there are connections between the left leg and right leg sLSTMs, and left arm and right arm sLSTMs, for capturing strong symmetrical dependences between the corresponding body parts. Note that in V5 when the sLSTM for the left leg is unfolded, it only has an incoming connection from the left-leg sLSTM from the previous time step, and likewise for all the other limb sLSTMs. In contrast, in V6, for the left-leg and right-leg sLSTMs, when the sLSTMs are unfolded, the left-leg sLSTM has an incoming connection from both the left-leg sLSTM and the right-leg sLSTM from the previous time step, and likewise for the right-leg sLSTM. Variant V6 tests if adding connections between the sLSTMs representing symmetrical body parts improves performance relative to V5.

**Network Details:** In V1, the multilayer perceptron consists of three fully-connected layers, the first and second ones with 1024 hidden units, and the last one with 3K units. In V2-V6, the sLSTMs have hidden units of size 128, and a memory size of 3. The sLSTMs have hidden units of size 1024, and a memory size of 3K. Whenever there is a connection from multiple sources into an LSTM, these outputs are concatenated for input into the LSTM.

### 5.4 Experiments

#### 5.4.1 Setup

**Datasets.** We evaluate human 3D pose estimation on the HumanEva-I [136] and Human3.6M datasets [56]. HumanEva-I and Human3.6M contain video clips each ranging from 10 seconds to 2 minutes. The clips are synchronized with ground-truth 3D body pose data from a motion capture system. HumanEva-I contains 4 subjects performing 6 different action classes. Human3.6M contains 7 subjects performing 15 different action classes. While Human3.6M also contains depth data and additional metadata, we only use video and ground-truth 3D body poses in our experiments.

**Evaluation Setup.** For HumanEva-I, we follow the standard protocol [76, 168] by evaluating on “Walking” and “Jogging” from subjects S1, S2 and S3.

For Human3.6M, it appears that the literature reports results on different train and test splits. We summarize the two protocols below. For completeness, we evaluate using both protocols in our experiments.
Table 5.1: Comparison of State-of-the-Art Approaches on HumanEva-I (MPJPE)

<table>
<thead>
<tr>
<th>Method</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>Radwan 2013 [115]</td>
<td>89.5</td>
</tr>
<tr>
<td>Wang 2014 [152]</td>
<td>71.3</td>
</tr>
<tr>
<td>Simo-Serra 2013 [137]</td>
<td>56.7</td>
</tr>
<tr>
<td>Bo 2010 [12]</td>
<td>48.7</td>
</tr>
<tr>
<td>Kostrikov 2014 [76]</td>
<td>40.3</td>
</tr>
<tr>
<td>Yasin 2016 [168]</td>
<td>38.9</td>
</tr>
<tr>
<td>V1</td>
<td>56.3</td>
</tr>
<tr>
<td>V2</td>
<td>54.6</td>
</tr>
<tr>
<td>V3</td>
<td>48.2</td>
</tr>
<tr>
<td>V4</td>
<td>42.7</td>
</tr>
<tr>
<td>V5</td>
<td>37.8</td>
</tr>
<tr>
<td>V6</td>
<td>35.3</td>
</tr>
</tbody>
</table>

Table 5.2: Comparison of State-of-the-Art Approaches on Human3.6M (MPJPE)

<table>
<thead>
<tr>
<th>Method</th>
<th>Protocol 1 Error</th>
<th>Protocol 2 Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yasin 2016</td>
<td>108.3</td>
<td>-</td>
</tr>
<tr>
<td>Rogez 2016</td>
<td>88.1</td>
<td>121.2</td>
</tr>
<tr>
<td>Zhou 2016</td>
<td>-</td>
<td>113.01</td>
</tr>
<tr>
<td>Tekin 2016</td>
<td>-</td>
<td>124.97</td>
</tr>
<tr>
<td>V1</td>
<td>120.82</td>
<td>133.63</td>
</tr>
<tr>
<td>V2</td>
<td>116.55</td>
<td>125.03</td>
</tr>
<tr>
<td>V3</td>
<td>111.14</td>
<td>123.22</td>
</tr>
<tr>
<td>V4</td>
<td>108.27</td>
<td>119.89</td>
</tr>
<tr>
<td>V5</td>
<td>87.96</td>
<td>112.97</td>
</tr>
<tr>
<td>V6</td>
<td><strong>81.13</strong></td>
<td><strong>105.65</strong></td>
</tr>
</tbody>
</table>

1. **Protocol 1.** In [168, 76, 122], the entire dataset is partitioned into six training subjects (S1, S5, S6, S7, S8, S9) and one test subject (S11). Evaluation is performed on every 64th frame of S11’s video clips.
Table 5.3: Per Class Evaluation of Human3.6M by Protocol 1 (MPJPE)

<table>
<thead>
<tr>
<th>Method</th>
<th>Direction</th>
<th>Discuss</th>
<th>Eat</th>
<th>Greet</th>
<th>Phone</th>
<th>Pose</th>
<th>Purchase</th>
<th>Sit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yasin 2016 [168]</td>
<td>88.4</td>
<td>72.5</td>
<td>108.5</td>
<td>110.2</td>
<td>97.1</td>
<td>81.6</td>
<td>107.2</td>
<td>119.0</td>
</tr>
<tr>
<td>Rogez 2016 [122]</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Ours</td>
<td><strong>80.88</strong></td>
<td><strong>63.56</strong></td>
<td><strong>98.92</strong></td>
<td><strong>102.15</strong></td>
<td><strong>88.18</strong></td>
<td><strong>72.37</strong></td>
<td><strong>100.01</strong></td>
<td><strong>110.03</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Method</th>
<th>SitDown</th>
<th>Smoke</th>
<th>Photo</th>
<th>Wait</th>
<th>Walk</th>
<th>WalkDog</th>
<th>WalkPair</th>
<th>Avg</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yasin 2016 [168]</td>
<td>170.8</td>
<td>108.2</td>
<td>142.5</td>
<td>86.9</td>
<td>92.1</td>
<td>165.7</td>
<td>102.0</td>
<td>108.3</td>
</tr>
<tr>
<td>Rogez 2016 [122]</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>88.1</td>
</tr>
<tr>
<td>Ours</td>
<td><strong>162.63</strong></td>
<td><strong>99.87</strong></td>
<td><strong>136.06</strong></td>
<td><strong>80.34</strong></td>
<td><strong>85.17</strong></td>
<td><strong>159.15</strong></td>
<td><strong>92.74</strong></td>
<td><strong>81.13</strong></td>
</tr>
</tbody>
</table>

Table 5.4: Per Class Evaluation of Human3.6M by Protocol 2 (MPJPE)

<table>
<thead>
<tr>
<th>Method</th>
<th>Direction</th>
<th>Discuss</th>
<th>Eat</th>
<th>Greet</th>
<th>Phone</th>
<th>Pose</th>
<th>Purchase</th>
<th>Sit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zhou 2016 [179]</td>
<td>87.36</td>
<td>109.31</td>
<td>87.05</td>
<td>103.16</td>
<td>116.18</td>
<td>106.88</td>
<td>99.78</td>
<td>124.52</td>
</tr>
<tr>
<td>Tekin 2016 [146]</td>
<td>102.41</td>
<td>147.72</td>
<td>88.83</td>
<td>125.38</td>
<td>118.02</td>
<td>112.38</td>
<td>129.17</td>
<td>138.89</td>
</tr>
<tr>
<td>Ours</td>
<td><strong>77.78</strong></td>
<td><strong>102.40</strong></td>
<td><strong>78.91</strong></td>
<td><strong>93.37</strong></td>
<td><strong>109.36</strong></td>
<td><strong>97.43</strong></td>
<td><strong>90.21</strong></td>
<td><strong>116.22</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Method</th>
<th>SitDown</th>
<th>Smoke</th>
<th>Photo</th>
<th>Wait</th>
<th>Walk</th>
<th>WalkDog</th>
<th>WalkPair</th>
<th>Avg</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zhou 2016 [179]</td>
<td>199.23</td>
<td>107.42</td>
<td>139.46</td>
<td>118.09</td>
<td>79.39</td>
<td>114.23</td>
<td>97.70</td>
<td>113.01</td>
</tr>
<tr>
<td>Tekin 2016 [146]</td>
<td>224.9</td>
<td>118.42</td>
<td>182.73</td>
<td>138.75</td>
<td>55.07</td>
<td>126.29</td>
<td><strong>65.76</strong></td>
<td>124.97</td>
</tr>
<tr>
<td>Ours</td>
<td><strong>193.20</strong></td>
<td><strong>98.29</strong></td>
<td><strong>130.28</strong></td>
<td><strong>109.47</strong></td>
<td>71.35</td>
<td><strong>104.39</strong></td>
<td>89.61</td>
<td><strong>105.65</strong></td>
</tr>
</tbody>
</table>

Table 5.5: Training Times for Variants on Human3.6M

<table>
<thead>
<tr>
<th>Variant</th>
<th>Time (hrs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>V1</td>
<td>33.1</td>
</tr>
<tr>
<td>V2</td>
<td>37.2</td>
</tr>
<tr>
<td>V3</td>
<td>30.4</td>
</tr>
<tr>
<td>V4</td>
<td>53.8</td>
</tr>
<tr>
<td>V5</td>
<td>44.4</td>
</tr>
<tr>
<td>V6</td>
<td>45.3</td>
</tr>
</tbody>
</table>
2. **Protocol 2.** In [179, 146, 91], the entire dataset is partitioned into five training subjects (S1, S5, S6, S7, S8) and two test subjects (S9, S11). The setup of [179] downsamples the videos from 50 fps to 10 fps.

**Training Details.** We train using the Adam optimizer with the default hyperparameters. For Human3.6M, the network was trained for 15 epochs. For HumanEva-I, due to the significantly smaller training set size, the network was trained for 25 epochs. The network was run on Tesla K80 GPUs.

**Metrics.** As is done in the literature, our metric for estimating 3D pose from an image is mean per joint position error (MPJPE), which is the mean Euclidean distance between the groundtruth and predicted joint locations.

### 5.4.2 Quantitative Results

Table 5.1 compares our results with previously reported results on HumanEva-I Dataset. As we can see, our best performing variant is V6. Our best performing variant also outperforms all other published approaches.

Table 5.2 compares our results with previously reported on Human3.6M Dataset. We compared using the two protocols as previously described. We see that for both protocols, the best performing variant is V6. Our best performing variant also outperforms all other published approaches, which have arguably higher complexity. Compared to HumanEva-I dataset, our approach evaluated on Human3.6M outperforms by a larger margin than the other published results. This is reasonable because Human3.6M has more data.

Table 5.3 compares our per-scenario results with prior work on Human3.6M using Protocol 1. Table 5.4 compares our per-scenario results with prior work on Human3.6M using Protocol 2. These results are generally reasonable: scenarios with more self-occlusion are more difficult. For instance, scenarios where the subject is standing up generally have lower error than ones where the subject is sitting down. In fact, the more the pose deviates from the “standard” pose of standing up or walking, the higher the error.

Table 5.5 presents some training times for the different variants. We can see the trade-off of complexity versus time: generally the more LSTM units and complex connections, the longer the training times. While V6 is generally slower than almost all the other variants, it still yields the best accuracy of them all.
From the results, we can conclude that the best variant for our approach is V6. This makes sense, because joints on limbs are conditioned on other joints as well as on the corresponding limb (e.g. left and right arms), and all limbs are conditioned on the spine location. The variant V6 also incorporates all the elements we sought out: (i) conditioned on time, (ii) conditioned based on a graphical model of the skeleton, and (iii) joints on limbs conditioned on other limb joints and the corresponding symmetrical limb.
5.4.3 Qualitative Results

For qualitative results, we present some poses from experiments with the Human3.6M Dataset. Figure 5.4 presents some results for human 3D pose estimation from images. We see that the 3D pose predictions are only slightly off for the cases that perform well. For the cases that do not perform well, the predicted poses are still reasonable despite a few wrong details. For instance, in Figure 5.4, for bad example 5, the knee is crossed the other way and the arms are lifted in
the predicted output. This is arguably reasonable because at least one knee is still crossed. For bad example 6, both hands are on the knees rather than just one hand in the predicted output. This is perhaps reasonable because our winning architecture favors symmetry, yet we still see correct performances in good example 2 for instance. Finally, both bad cases are a result of self occlusion so predicting poses for these is challenging.

Figure 5.5 visualizes some samples from the latent distribution. We can see that the latent distribution was able to capture a variety of realistic poses, from sitting down to standing up and gesturing.

GANs have been demonstrated to yield a smooth latent representation. We show that our GAN has this property in Figure 5.6. We sampled two poses from our latent distribution, $z_1$ and $z_2$. Figure 5.6 demonstrates that sampling the interpolation between $z_1$ and $z_2$ yields smooth poses, suggesting a smooth latent representation has been learned by GAN. This also makes sense because our joints are conditioned on previous joints in our generator architecture.

5.5 Summary

We presented a generative framework using GANs for human 3D pose estimation that achieves competitive results without resorting to additional data augmentation. We proposed a novel generator that uses part-based modeling with LSTMs. An ablation study was performed to determine the best generator architecture. We verified our hypothesis that GAN training does yield a smooth latent space of 3D poses. Overall, we demonstrated that our GAN architecture is effective for human 3D pose estimation.
Chapter 6: Explaining Estimation of Human 3D Poses by Disentangling Generative Adversarial Networks

6.1 Introduction

In this chapter, we present another framework for human 3D pose estimation, where again predicting specific joint locations is fine-grained recognition of joints as parts from images. In contrast to Chapter 5, we present an approach that achieves better results and has the capability to give explanations when making predictions. Here, we motivate again our problem statement.

Predicting 3D locations of human body joints from RGB images (a.k.a. 3D pose estimation) is a longstanding problem in computer vision [52, 107, 32, 101] with a variety of applications including surveillance, human-computer interaction, motion capture and virtual reality. This is a challenging problem due to large variations of body part layouts and self-occlusions. To address these challenges, some previous methods resort to additional data beyond 2D images such as multiple viewpoints [4, 51] and depth [116, 169, 134]. This chapter focuses on recovering 3D poses from images only.

With the recent advent of large datasets with detailed annotations of human poses [56], there has been an increased interest in developing deep architectures for 3D pose estimation. These approaches can be generally divided into two frameworks: (i) discriminative [2, 12, 64, 123, 148], where the pose is directly regressed from image data, and (ii) generative [35, 135, 149], where the pose space of plausible skeleton configurations is searched to align with image data. The former typically use multiple streams of processing, resulting in arguably very complex deep models that may not be easy to train. The latter usually search for the best matching 3D pose from a library of exemplars to the pose extracted from the image and hence have difficulties to generalize to new 3D poses beyond the exemplars.

In this chapter, we approach human 3D pose estimation from the generative adversarial framework. A Generative Adversarial Network (GAN) consists of two competing subnetworks: i) a ‘generator’ network which generates data from an unknown distribution, and ii) a ‘discriminator’ network which discriminates between the generated samples and the ones from true observations. The goal of learning a GAN is to estimate a generator that models the true data dis-
Figure 6.1: Overview of Our Explainable GAN Model

GANs have been recently demonstrated as state-of-the-art performers for a wide range of computer vision problems [42, 114, 128, 120, 97, 150, 37]. However, to the best of our knowledge, GANs have not been used for 3D pose estimation.

As shown in Figure 6.1, given an image, we use a GAN to project pixels onto a latent embedding space, and then pass the resulting feature vector to the generator subnetwork for structured prediction of 3D locations of human body joints.

GANs are suitable for our purposes for several reasons. First, they have been shown to learn a smooth latent representation from training data [42], which helps with generalization to new 3D poses. Second, their learning formulation helps alleviate overfitting, even when training data may not uniformly represent all possible 3D poses.

Our key contribution is adding interpretability to the GAN’s latent embedding space, which makes it possible to provide explanations about the GAN’s 3D pose estimation. Our motivation stems from an observation that the literature presents scarce theoretical underpinnings of GANs, despite their recent success in a variety of vision problems. One reason could be the inherent difficulty of explaining the GAN’s latent feature space. A better understanding of this latent space seems critical, as it directly affects the GAN’s structured predictions.

As a step toward increasing GAN interpretability, we seek to disentangle the factors of variation in the latent space. Specifically, we model a latent clustering of features in the embedding space, which naturally arise from poses that people frequently assume (e.g., sitting yoga pose, walking pose). These clusters can be estimated from training data in a supervised or unsuper-
vised manner. For this, we combine the GAN with a variational autoencoder (VAE) [88] that uses a mixture of Gaussians as a prior distribution of the GAN’s latent features. The Gaussian mixture components serve to encode canonical human poses seen in training data. In this way, we enable the GAN to explain its predictions in terms of the dominant mixture components (i.e. canonical poses) identified in the embedding process of the input image.

From above, we specify a novel GAN training procedure that augments the standard adversarial discriminative loss with an additional “interpretability” regularization aimed at disentangling the factors of variation in the latent space. The key novelty of our training method is a new way to stabilize learning under the two learning objectives. Inspired by Karras et al. [66], instead of progressively adding layers during training, we progressively backpropagate loss from increasing subsets of the GAN’s output. Initially, we evaluate loss only on a small, select number of body-joint predictions, and then progressively add more joints until we obtain full 3D pose estimation.

Our main contributions include the following:

1. Modeling human 3D poses using GANs. We are not aware of prior work that uses GANs for this purpose.
2. Extending VAE with GANs for encoding images using a mixture of Gaussians as a prior distribution for generating 3D poses.
3. Regularizing GANs to promote disentangling factors of variation in the latent space, and thereby affording explanations during inference.
4. Stabilizing GAN training by progressive addition of outputs in the loss function.

In the following sections, Section 6.2 places our approach in the context of prior work; Section 6.3 specifies our model and learning; and Section 6.4 presents our results.

6.2 Literature Review

**Generative Adversarial Networks.** GANs have made significant advances in image modeling and understanding [42, 114, 128, 120]. Recently, GANs have been also applied to generating images from semantic segmentation [57]. GANs have also been combined with variational autoencoders (VAE) [88] for recognizing 3D shapes [157]. The work of Gurumurthy et al. [45] is closest to ours, introducing a Gaussian mixture model for GANs. However, we are not aware of any prior work that uses GANs for modeling or estimating human 3D poses. To our knowledge, there is no prior work on explaining GAN’s predictions using a Gaussian mixture.
**Disentangling Factors of Variation.** Recent work seeks to factorize deep feature spaces—the problem known as disentanglement—toward improving training efficiency and overall performance [33, 98]. For example, Fu et al. [33] combine a GAN and VAE for cross-domain disentanglement. Mathieu et al. [98] introduce a conditional generative model, where a GAN is used for identifying latent factors in the deep feature space. In our work, we also combine a GAN and VAE but with the following differences: (i) we use a Gaussian mixture model for disentanglement and (ii) we regularize Gaussian components in the mixture to model well the canonical human poses found in training images. In other words, we identify semantically meaningful factors of variation and in this way afford providing explanations of GAN’s predictions.

**Human 3D Pose Estimation.** A majority of previous deep learning approaches formulate the problem of 3D pose estimation as regression from 2D image to 3D pose. Li et al. [91] train a multi-task regression convolutional neural network to predict 3D pose directly from images. Tekin et al. [146] work with spatiotemporal features of an image sequence to learn a regression model. More recently, Zhou et al. [180] present a weakly-supervised approach that uses 2D images of human poses in-the-wild to regress to 3D human pose. In contrast to these and other regression approaches, we take a generative approach.

Exemplar-based methods for 3D pose estimation have also been explored [133]. One of the main challenges of exemplar-based approaches is generalization to novel poses outside of the training set. Jiang et al. [61] address generalization by matching upper and lower bodies individually. Yasin et al. [168] use an energy minimization approach to adapt exemplars to better match image measurements. Rogez et al. [122] synthesize new 2D images through image-based rendering. Other approaches warp from 2D image descriptors to 3D exemplars, such as shape context [2, 102] or silhouette features [55]. In our work, we train a generative model that learns a dictionary of exemplars that not only come from a smooth distribution [42] but the exemplars can provide explanations of inferred pose estimates.

### 6.3 Technical Approach

Figure 6.2 illustrates our model. We first train a GAN on human 3D poses to generate novel poses \( \hat{x} \) based on explainable canonical poses. We then train VAE to project input images \( y \) onto the latent space of human 3D poses \( z \), parameterized by a Gaussian mixture model. During test time, an input image \( y \) is mapped to the 3D-pose latent representation \( z = E(y) \). The 3D pose is then recovered through the generator \( \hat{x} = G(z) \). The latent space \( z \) is interpretable because of the
regularization (specified in Section 6.3.3). The bottom row visualizes an example of the learned means $\mu_i$ of the Gaussian mixture components.

As Figure 6.2 illustrates, our model represents a standard combination of a GAN with a VAE, but with a key difference from prior work that the latent feature space of the VAE is parameterized by a Gaussian mixture model. For brevity, we refer to our model as GAN. Our GAN is regularized with an “interpretability objective” so that each Gaussian component is associated with semantically meaningful patterns in training data. In this section, we review GANs and VAEs, and specify how we merge them along with our interpretability objective.

6.3.1 Generative Adversarial Network

A GAN is a neural network that consists of two competing subnetworks: (i) a generator network $G$ which generates data from an unknown distribution (e.g. ground-truth 3D poses) and (ii) a discriminator network $D$ which discriminates between the generated samples from $G$ and the ones from the unknown true distribution. The goal of learning a GAN is to find a generator $G$ which models the true data distribution while maximizing the probability of the discriminator
making a mistake.

Let \( x \) denote a true data sample, \( \mathbf{z} \sim p_{\mathbf{z}}(\mathbf{z}) \) denote a randomly sampled vector from a prior distribution \( p_{\mathbf{z}}(\mathbf{z}) \), and \( \hat{x} = G(\mathbf{z}) \) indicate the corresponding generated sample. The ground-truth 3D pose, \( x \), is parameterized as a vector of \( K = 3J \) elements, where \( J \) is the number of joints in a human skeleton. The elements of \( x \) are normalized in the range of \([-1, 1]\). We interpret each 3D point as an offset from a reference 3D point, where the spherical coordinate system is used to parameterize the offsets \((r, \theta, \phi)\). For instance, the left elbow is parameterized as an offset to the left shoulder. These offsets form a tree, whose root represents the hip. The root has no offset. In this paper, \( \mathbf{z} \) is a 50-dimensional vector, and its sampling scheme is specified in Section 6.3.2. The space of \( \mathbf{z} \) vectors sampled from \( p_{\mathbf{z}}(\mathbf{z}) \) is referred to as the latent space.

Our architecture of \( G \) represents a series of fully connected layers (specifically, 3 layers) where the input accepts \( \mathbf{z} \) and outputs a \( K \)-dimensional vector with \( \tanh \) nonlinearity. Our discriminator \( D \) mirrors \( G \)'s architecture with a final output from a sigmoid layer that yields a probability estimate of whether the input is “true/real” or “generated/fake”.

Learning a GAN is specified as minimizing the following loss function:

\[
L_{\text{GAN}} = \log D(x) + \log (1 - D(\hat{x})).
\] (6.1)

\( D \) is trained to maximize the probability of correct sample classification—“true” vs. “generated”—and simultaneously \( G \) is trained to minimize \( \log(1 - D(\hat{x})) \).

6.3.2 Mixture of Gaussians

Prior work typically makes the assumption that the \( \mathbf{z} \) vector is sampled from a standard Gaussian distribution in the \( K \)-dimensional latent space. In our approach, \( \mathbf{z} \) is sampled from a Gaussian mixture:

\[
p_{\mathbf{z}}(\mathbf{z}) = \sum_{i=1}^{N} \phi_i g(\mathbf{z} | \mu_i, \Sigma_i),
\] (6.2)

where \( \phi_i \)'s are assumed uniform weights of the mixture components, \( \phi_i = 1/N \), and \( g(\mathbf{z} | \mu_i, \Sigma_i) \) represents the Gaussian distribution with \( K \)-dimensional mean \( \mu_i \) and diagonal covariance matrix \( \Sigma_i = \text{diag}(\sigma_i^2) = \text{diag}([\sigma_{i1}^2, \sigma_{i2}^2, \ldots, \sigma_{iK}^2]). \)

For generating “fake” samples in the GAN, we choose one of the \( N \) Gaussian components according to \( \phi_i \), sample the latent vector \( \mathbf{z} \) from the selected \( i \)th Gaussian, and generate \( \hat{x} = G(\mathbf{z}) \).
This allows us to efficiently learn all of the Gaussian parameters \(\{\mu_i\}\) and \(\{\sigma_i\}\), \(G\), and \(D\) using the aforementioned standard GAN training on “fake” and “real” examples.

Efficient learning of \(\{\mu_i\}\) and \(\{\sigma_i\}\), along with all of the other GAN parameters, is made feasible by following the “reparameterization” method, described in [69, 45], where \(z\) sampled from \(i\)-th Gaussian component of the mixture can be expressed as

\[
    z = \mu_i + \sigma_i \odot \epsilon,
\]

where \(\odot\) is the symbol for element-wise product, and \(\epsilon \sim \mathcal{N}(0, I)\) is a K-dimensional vector sampled from a normal distribution with zero mean and identity covariance matrix. From (6.3), it follows that \(\{\mu_i\}\) and \(\{\sigma_i\}\) represent a subset of GAN weights which can be efficiently learned via backpropagation along with the other weights.

To initialize learning of \(\mu_i\), we sample from a uniform distribution \(\mathcal{U}(-1, 1)\) along all \(K\) dimensions. As initial \(\sigma_i\), we use a small, non-zero value (specifically, 0.1) for all \(K\) dimensions of \(\sigma_i\).

Note that since \(\sigma_i\)’s can collapse to zero in training, we add the following regularization over the \(N\) mixture components (c.f., [45]):

\[
    L_{\text{reg}} = \frac{1}{N} \sum_{i=1}^{N} \|1 - \sigma_i\|^2.
\]

6.3.3 Interpretability

We introduce a new interpretability loss, \(L_{\text{explain}}\), in our GAN formulation. Interpretability loss learns to associate each Gaussian component of the mixture in the latent space with a pre-specified concept. In our setting, the concepts represent canonical human poses with semantic meaning (e.g. standing, sitting). These canonical poses can be extracted from training data in an unsupervised manner (e.g. using K-means clustering) or in a supervised manner by resorting to ground-truth annotations of pose classes in training data.

What we would like to enforce is that each Gaussian component learns a canonical human pose. Consequently, samples \(z\) around the mean points \(\mu_i\) of the mixture in the latent space correspond to 3D variations of the respective canonical poses. Note that our pre-specified concepts should lie in the space of \(x\), not in the latent space, since \(x\)’s directly represent 3D locations of human-body joints. Therefore, the goal of learning is to estimate parameters of the Gaussian
mixture in the latent space whose samples \( z \) will be mapped through the generator \( \hat{x} = G(z) \) to the corresponding canonical poses \( x_i^{\text{canonical}} \), \( i \in \{1, \ldots, N\} \).

For this learning, we specify the following interpretability loss

\[
L_{\text{explain}}(i) = \exp(-\lambda \|\mu_i - z\|_2^2) \|x_i^{\text{canonical}} - \hat{x}\|_2^2,
\]

(6.5)

where \( x_i^{\text{canonical}} \) is the \( i \)-th canonical pose, \( i \in \{1, \ldots, N\} \), mined from training data, and \( \lambda \) is a scale parameter. Note that we penalize \( \|x_i^{\text{canonical}} - \hat{x}\|_2 \) the most when the latent vector \( z \) is equal to \( \mu_i \), essentially the case when we want the generated \( \hat{x} \) to be the canonical pose \( x_i^{\text{canonical}} \).

### 6.3.4 Variational Autoencoder

As in prior work \([157, 88]\), we use a VAE to embed the input image \( y \) onto the latent space, resulting in the vector \( z \) governed by the underlying distribution \( p_z(z) \). In turn, \( z \) can be passed to the GAN to generate predictions \( \hat{x} = G(z) \).

VAE consists of encoder \( E \) and generator \( G \) networks, aimed at modeling a posterior distribution \( p(y|z) \) of the observed data, \( y \), given an unobserved latent variable, \( z \), where \( z = E(y) \) and \( \hat{x} = G(E(y)) \). Let \( z \sim p_z(z) \) be the unobserved latent variable sampled from distribution \( p_z(z) \). The learning goal for the VAE is to minimize a loss function, \( L_{\text{VAE}} \), defined in terms of the KL divergence, \( D_{\text{KL}} \), between the variational distribution \( q(z|y) \) and the prior \( p_z(z) \), as well as a reconstruction error between the output and the ground-truth \( \|G(E(y)) - x\|_2^2 \):

\[
L_{\text{VAE}} = \|G(E(y)) - x\|_2^2 + D_{\text{KL}}(q(z|y)||p(z)).
\]

(6.6)

The second term in (6.6) ensures that the generator \( G \) can sample \( z \) from a distribution conditioned on the input image \( y \) as similar as possible to \( p_z(z) \).

(The astute reader will again note that our formulation is technically not an autoencoder since \( G(E(y)) = \hat{x} \neq \hat{y} \) where \( \hat{y} \) would be a reconstructed image from \( y \). While this may be the case, the literature still refers to this as a VAE in some cases \([157]\), thus we will put this technicality away as a side note.)

Our encoder is based on the architecture of Pavlakos et al. \([108]\). It is a series of multiple fully convolutional components that alternate between decreasing and increasing resolution like a series of hourglasses. Instead of outputting a dense 3D volume with separate per voxel likelihoods for each joint, we have an additional series of convolutions at the end that output \( \varepsilon \) and \( s \),
where $\epsilon$ has the same dimensionality as $z$ and $s$ is an $N$-dimensional softmax output. $\epsilon$ and $s$ are combined with $\mu_i$ and $\sigma_i$ to form $z$ as

$$z = \sum_{i=1}^{N} s_i (\mu_i + \sigma_i \odot \epsilon)$$ (6.7)

The resulting $z$ is then passed to the generator $G$ to produce the pose $\hat{x}$.

### 6.3.5 Training

We combine GAN and VAE, and train using the following loss function that incorporates the aforementioned losses of GAN, VAE, and interpretability:

$$L = L_{GAN} + \lambda_1 L_{reg} + \lambda_2 L_{explain} + \lambda_3 L_{VAE}$$ (6.8)

where $\lambda$’s are hyperparameters for weighting each loss in (6.8).

We first train the GAN with the interpretability loss to produce a generative model of 3D poses. The VAE is then trained to encode images into poses.

To improve the stability of GAN training, we introduce a new progressive training strategy, motivated by the approach of Karras et al. [66] aimed at a different problem, that of image generation. Karras et al. progressively grow GANs by adding more layers during training so that earlier layers can stabilize first. By focusing the learning process on smaller resolutions at first, their technique enables image generation at very high resolutions.

The key difference from their approach is that instead of growing the layers of the GAN, we...
progressively add more outputs to the GAN. This effectively amounts to accounting for increasingly more predictions of joints of a human skeleton in the loss function, given by (6.8), starting from a few joints. Figure 6.3 provides an illustration of our progressive training. Intuitively, we train the GAN to generate the spine of a 3D human pose first, then generate a limb, then another limb, etc. as the number of epochs increase. During training of the spine, for instance, we ignore the other joints in $G$, in $D$, and in the loss functions. We empirically find that learning in the following order gives the best performance: spine, right leg, left leg, right arm, and left arm.

6.4 Experiments

6.4.1 Setup

Datasets. We evaluate human 3D pose estimation on the HumanEva-I [136] and Human3.6M datasets [56]. HumanEva-I and Human3.6M contain video clips each ranging from 10 seconds to 2 minutes. The clips are synchronized with ground-truth 3D body pose data from a motion capture system. HumanEva-I contains 4 subjects, performing 6 different action classes. Human3.6M contains 7 subjects, performing 15 different action classes. While Human3.6M also contains depth data and additional metadata, we only use video, ground-truth 3D body poses and action class labels in our experiments.

To the best of our knowledge, we are unaware of other large scale datasets with 3D pose annotations. Since HumanEva-I and Human3.6M Datasets contain similar indoor environments, we also evaluate on another dataset to demonstrate that our approach can transfer reasonably well to in-the-wild images. Since datasets with 3D groundtruth annotations are currently limited, we choose a human 2D pose estimation dataset, specifically MPII dataset. MPII is widely used for evaluation for 2D pose estimation. Since it is unfair to evaluate our method on MPII due to different levels of supervision (i.e., 2D vs. 3D) and since 3D annotations are unavailable for evaluation, we will instead present qualitative results (c.f., [147]).

Evaluation Setup. For HumanEva-I, we follow the standard protocol [76, 168] by evaluating on “Walking” and “Jogging” from subjects S1, S2 and S3. For Human3.6M, we follow the setup of [179, 146, 91], where the entire dataset is partitioned into five training subjects (S1, S5, S6, S7, S8) and two test subjects (S9, S11). The setup of [179] also downsamples the videos from 50 fps to 10 fps.

Training Details. For training, we perform standard data augmentation by flipping horizon-
tally and randomly cropping. We train using the Adam optimizer with the default hyperparameters. For Human3.6M, the network was trained for 15 epochs. For HumanEva-I, due to the significantly smaller training set size, the network was trained for 25 epochs. The network was run on Tesla K80 GPUs.

**Metrics.** As is done in the literature, our metric for estimating 3D pose from an image is mean per joint position error (MPJPE), which is the mean Euclidean distance between the groundtruth and predicted joint locations.

### 6.4.2 Baselines

We define the following baselines. In all baselines, we have a Gaussian mixture model with $N$ components.

- **B1. No Explanations:** In this baseline, we omit the interpretability regularization term in our loss function. Therefore, this baseline is just a GAN with a mixture of Gaussians model for the latent variable.

- **B2. Labeled Explanations:** In this setup, the 3D pose ground truth data is first clustered for each label (i.e., action class) using K-means, yielding $k$ labeled clusters for each action class. This yields $K = Ck$ clusters where $C$ is the number of action classes. Instead of clustering every action class, we only cluster some “representative” poses: standing/walking, sitting and crouching. We do this because poses from some action classes look very similar to poses from another action class (e.g., walking and walking together look similar). This is also done in Mehta et al. [99]. We then incorporate the interpretability regularization in our training where each Gaussian component for the latent variable is associated with a cluster. We set $N = K + 1$, where the last component is free to learn anything.

- **B3. Unlabeled Explanations:** This setup is similar to B2, except the 3D poses across all classes are clustered together. In other words, no class labels are used, or alternatively, every action class in B2 is collapsed into one class.

### 6.4.3 Quantitative Results

Table 6.1 compares our results with previously reported results on HumanEva-I Dataset. We see that our best performing baseline is B3, followed barely by B2, which also outperforms all other published approaches.
Table 6.1: HumanEva-I Quantitative Evaluation (MPJPE)

<table>
<thead>
<tr>
<th>Method</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>Radwan 2013 [115]</td>
<td>89.5</td>
</tr>
<tr>
<td>Wang 2014 [152]</td>
<td>71.3</td>
</tr>
<tr>
<td>Simo-Serra 2013 [137]</td>
<td>56.7</td>
</tr>
<tr>
<td>Bo 2010 [12]</td>
<td>48.7</td>
</tr>
<tr>
<td>Kostrikov 2014 [76]</td>
<td>40.3</td>
</tr>
<tr>
<td>Yasin 2016 [168]</td>
<td>38.9</td>
</tr>
<tr>
<td>Pavlakos 2017 [108]</td>
<td>24.3</td>
</tr>
<tr>
<td>Ours (B1)</td>
<td>35.4</td>
</tr>
<tr>
<td>Ours (B2)</td>
<td>23.6</td>
</tr>
<tr>
<td>Ours (B3)</td>
<td>23.2</td>
</tr>
</tbody>
</table>

Table 6.2 compares our results with those previously reported on Human3.6M Dataset. We see that the best performing baseline is B3, followed barely by B2. Our best performing variant also outperforms all other published approaches. In Table 6.2, we also compare our per-scenario results with prior work on Human3.6M. These results are generally reasonable: scenarios where more self-occlusion is present makes pose estimation more difficult.

We also study the number of Gaussian components $N$. Figure 6.4 shows the results of running B1 on Human36.M for different values of $N$. We observe that larger values of $N$ improve our accuracy, which suggests that our model may learn more diverse clusters. We can also see that there is a stable region over a range of values, suggesting that the choice of $N$ is not highly sensitive around a particular region. Finally, increasing $N$ beyond a certain point does not improve our results, which may suggest overfitting. In the end, we chose to report our results using $N = 75$.

From our overall results, we conclude that the best baseline is B3, followed very closely by B2. This is plausible since in B2, we manually choose clusters, whereas in B3 we let the clustering algorithm choose. Arguably, while choosing our clusters for B2 affords interpretability in a more controlled manner, it is inferior to an algorithm automatically choosing clusters. Nonetheless, the clusters chosen in B3 are reasonable, as we will see qualitatively later on. We can also conclude from our experiments that adding interpretability regularization improves results.
Table 6.2: Human3.6M Per Class Evaluation (MPJPE)

<table>
<thead>
<tr>
<th>Method</th>
<th>Direction</th>
<th>Discuss</th>
<th>Eat</th>
<th>Greet</th>
<th>Phone</th>
<th>Pose</th>
<th>Purchase</th>
<th>Sit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chen 2016 [17]</td>
<td>89.87</td>
<td>97.57</td>
<td>89.98</td>
<td>107.87</td>
<td>107.31</td>
<td>139.17</td>
<td>93.56</td>
<td>136.09</td>
</tr>
<tr>
<td>Tome 2017 [147]</td>
<td>64.98</td>
<td>73.47</td>
<td>76.82</td>
<td>86.43</td>
<td>86.28</td>
<td>110.67</td>
<td>68.93</td>
<td>74.79</td>
</tr>
<tr>
<td>Zhou 2016 [179]</td>
<td>87.36</td>
<td>109.31</td>
<td>87.05</td>
<td>103.16</td>
<td>116.18</td>
<td>110.68</td>
<td>99.78</td>
<td>124.52</td>
</tr>
<tr>
<td>Tekin 2016 [146]</td>
<td>102.41</td>
<td>147.72</td>
<td>88.83</td>
<td>125.38</td>
<td>118.02</td>
<td>112.38</td>
<td>129.17</td>
<td>138.89</td>
</tr>
<tr>
<td>Mehta 2016 [99]</td>
<td>59.69</td>
<td>69.74</td>
<td>60.55</td>
<td>68.77</td>
<td>76.36</td>
<td>85.42</td>
<td>59.05</td>
<td>75.04</td>
</tr>
<tr>
<td>Pavlakos 2017 [108]</td>
<td>58.55</td>
<td>64.56</td>
<td>63.66</td>
<td>62.43</td>
<td>66.93</td>
<td>70.74</td>
<td>57.72</td>
<td>62.51</td>
</tr>
<tr>
<td>Nie 2017 [158]</td>
<td>90.1</td>
<td>88.2</td>
<td>85.7</td>
<td>95.6</td>
<td>103.9</td>
<td>92.4</td>
<td>90.4</td>
<td>117.9</td>
</tr>
<tr>
<td>Zhou 2017 [180]</td>
<td>54.82</td>
<td>60.70</td>
<td>58.22</td>
<td>71.41</td>
<td>62.03</td>
<td>65.53</td>
<td>53.83</td>
<td>55.58</td>
</tr>
<tr>
<td>Ours (B1)</td>
<td>57.36</td>
<td>65.44</td>
<td>62.63</td>
<td>73.8</td>
<td>69.95</td>
<td>70.81</td>
<td>58.94</td>
<td>58.72</td>
</tr>
<tr>
<td>Ours (B2)</td>
<td>53.98</td>
<td>62.39</td>
<td>57.84</td>
<td>65.05</td>
<td>62.51</td>
<td>67.88</td>
<td>54.62</td>
<td>53.87</td>
</tr>
<tr>
<td>Ours (B3)</td>
<td><strong>53.87</strong></td>
<td><strong>60.17</strong></td>
<td><strong>56.60</strong></td>
<td><strong>63.32</strong></td>
<td><strong>61.45</strong></td>
<td><strong>65.32</strong></td>
<td><strong>52.71</strong></td>
<td><strong>53.65</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Method</th>
<th>SitDown</th>
<th>Smoke</th>
<th>Photo</th>
<th>Wait</th>
<th>Walk</th>
<th>WalkDog</th>
<th>WalkPair</th>
<th>Avg</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chen 2016 [17]</td>
<td>133.14</td>
<td>240.12</td>
<td>106.65</td>
<td>106.21</td>
<td>87.03</td>
<td>114.05</td>
<td>90.55</td>
<td>114.18</td>
</tr>
<tr>
<td>Tome 2017 [147]</td>
<td>110.19</td>
<td>172.91</td>
<td>84.95</td>
<td>85.78</td>
<td>86.26</td>
<td>71.36</td>
<td>73.14</td>
<td>88.39</td>
</tr>
<tr>
<td>Zhou 2016 [179]</td>
<td>199.23</td>
<td>107.42</td>
<td>139.46</td>
<td>118.09</td>
<td>79.39</td>
<td>114.23</td>
<td>97.70</td>
<td>113.01</td>
</tr>
<tr>
<td>Tekin 2016 [146]</td>
<td>224.9</td>
<td>118.42</td>
<td>182.73</td>
<td>138.75</td>
<td>55.07</td>
<td>126.29</td>
<td>65.76</td>
<td>124.97</td>
</tr>
<tr>
<td>Mehta 2016 [99]</td>
<td>96.19</td>
<td>122.92</td>
<td>70.82</td>
<td>68.45</td>
<td>54.41</td>
<td>82.03</td>
<td>59.79</td>
<td>74.14</td>
</tr>
<tr>
<td>Pavlakos 2017 [108]</td>
<td>76.84</td>
<td>103.48</td>
<td>65.73</td>
<td><strong>61.56</strong></td>
<td>67.55</td>
<td><strong>56.38</strong></td>
<td>59.47</td>
<td>66.92</td>
</tr>
<tr>
<td>Nie 2017 [158]</td>
<td>136.4</td>
<td><strong>98.5</strong></td>
<td>103.0</td>
<td>94.4</td>
<td>86.0</td>
<td>90.6</td>
<td>89.5</td>
<td>97.5</td>
</tr>
<tr>
<td>Zhou 2017 [180]</td>
<td>75.20</td>
<td>111.59</td>
<td><strong>64.15</strong></td>
<td>66.05</td>
<td>51.43</td>
<td>63.22</td>
<td>55.33</td>
<td>64.90</td>
</tr>
<tr>
<td>Ours (B1)</td>
<td>75.39</td>
<td>108.11</td>
<td>71.41</td>
<td>65.17</td>
<td>52.44</td>
<td>64.04</td>
<td>56.29</td>
<td>66.78</td>
</tr>
<tr>
<td>Ours (B2)</td>
<td>72.72</td>
<td>105.67</td>
<td>65.08</td>
<td>63.26</td>
<td>51.27</td>
<td>63.75</td>
<td>54.61</td>
<td>63.84</td>
</tr>
<tr>
<td>Ours (B3)</td>
<td><strong>71.76</strong></td>
<td>102.47</td>
<td>64.29</td>
<td>61.67</td>
<td><strong>49.29</strong></td>
<td>62.94</td>
<td><strong>53.96</strong></td>
<td><strong>62.91</strong></td>
</tr>
</tbody>
</table>

Finally, we demonstrate that our progressive training is essential in Table 6.3. We perform 10-folds of experiments on several baselines. The first baseline does not use progressive training. The second baseline chooses the joints randomly as the epochs increase. The third and final method is to choose the joints based on our specified ordering: spine, right leg, left leg, right arm, and left arm. We can see that our progressive training scheme, especially when using our sequential ordering, improves our performance. Our sequential ordering makes sense: from a training stability standpoint, the spine is usually upright or somewhat bent, making the problem easier to learn at first. Then the legs are learned, which is arguably easier to learn the arms since arms tend to exhibit more variation whereas legs are usually either straight or bent at 90 degrees (i.e., standing or sitting).
Table 6.3: Training Progression Baselines Study (MPJPE)

<table>
<thead>
<tr>
<th>Stabilization Method</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>99.13</td>
</tr>
<tr>
<td>Random Ordering</td>
<td>75.32</td>
</tr>
<tr>
<td>Sequential</td>
<td>66.78</td>
</tr>
</tbody>
</table>

6.4.4 Qualitative Results

For qualitative results, we present some poses from experiments with Human3.6M Dataset. Figure 6.5 presents some results for human 3D pose estimation from images. (From left-to-right and top-to-bottom, the first four examples are success cases and the last two examples are failure cases.) We see that the 3D pose predictions are only slightly off for the cases that perform well. For the cases that do not perform well, the predicted poses are still reasonable despite a few incorrect details. For instance, in Figure 6.5, for the first bad example, the knee is not fully crossed in the predicted output. For the second bad example, both hands are elevated rather than touching the ground in the predicted output. These predicted outputs are still arguably reasonable because the overall pose is correct. Finally, both bad cases are a result of self-occlusion, so predicting poses for these is challenging.
Figure 6.7 visualizes some clusters from the mixture of Gaussians latent distribution. Specifically, it visualizes cluster centers $\mu_i$. We chose some representative samples for each of the baselines. We can see that the latent distribution was able to capture a variety of realistic poses, from sitting down to standing up. Interestingly, the cluster centers for B2 and B3 look a bit more realistic than B1. This is plausible since the GAN can learn some realistic poses through the explanation regularization instead of learning from scratch.

We also demonstrate that our approach can transfer reasonably well to other domains. While it is unfair to evaluate our method on other datasets since they lack 3D annotations, we can nonetheless show that our approach can yield some reasonable results on other domains through qualitative evaluation (c.f., [147]). Figure 6.6 presents some qualitative results of human 3D pose predictions on MPII dataset. We can see that even with different backgrounds and body types,
Figure 6.7: Skeletons Generated from Means of Gaussian Components
we can get some reasonable results. However, we will not be able to generalize to exotic poses (e.g., sports with large limb articulations). Regardless, these results suggest that our architecture is promising if we have access to more 3D training data.

6.5 Summary

We presented a novel generative framework using GANs for human 3D pose estimation that not only outperforms the state of the art but also provides explanations of predictions. Our key contribution is a new way to disentangle factors of variation in a GAN and thus enable interpretability of its latent feature space, which in turn can be use to provide explanations of predictions. The disentanglement is formalized by parameterizing the latent space of GAN with a Gaussian mixture model. The Gaussian mixture is regularized such that its components represent canonical human poses seen in training data. We also introduced a new progressive training strategy that stabilizes GAN training and allows learning of human 3D poses to much higher accuracy. Finally, we demonstrated that adding the new interpretability objective in our GAN leads to performance improvements over the standard GANs and superior results relative to the state of the art on the benchmark HumanEva-I, Human3.6M, and MPII datasets, which also generalize to in-the-wild images.
Chapter 7: Conclusion

In this dissertation, we addressed the problem of fine-grained object recognition in images and videos where distinct object classes are visually very similar and/or access to training data is limited. We made a number of contributions.

First, we introduced two improvements to the HC-Search framework for structured prediction vision tasks: (1) a randomized segmentation space as a generic search space that leverages ultrametric contour map segmentations and (2) the application of DAGGER for training robust heuristic functions. We demonstrated that HC-Search with these improvements gives performance comparable to or better than the state of the art across three diverse vision tasks of labeling parts in images: semantic scene labeling, monocular depth estimation, and object detection (i.e. fine-grained recognition) in biological images. Our error decomposition analysis demonstrates that our improvements achieve significant performance boosts over previous attempts to apply HC-Search in computer vision. Since we worked with a limited number of biological images, we have demonstrated that our approach is effective for fine-grained recognition when training data is limited.

Second, we presented another search-based framework called HSnet Search with deep architectures that achieves competitive results for fine-grained recognition. HSnet is formulated with a built-in refinement mechanism to search for increasingly more informative parts and thus improve recognition, in addition to a robustness mechanism against wrongly identified parts during inference. Our search-based architecture consists of a search space defined on a CNN feature map, and our architecture parameterizes heuristic and successor functions by a new deep network architecture called HSnet. We specified two training settings: one where part location annotations are available and one where they are not available. The latter is addressed with a determinantal point process loss to obtain diverse proposals. Finally, our experimental results on Caltech-UCSD Birds 200-2011 and Cars-196 datasets demonstrated that sequential reasoning about object parts and removing background context are effective for fine-grained recognition. We demonstrated the validity of our hypothesis that robust fine-grained recognition is enabled by reasoning about discriminative image parts in a search-based framework.

Third, we addressed the problem of unsupervised video summarization, where the task of
picking a subset of key frames from a limited number of videos is fine-grained recognition of summary key frames from videos. We proposed a generative architecture based on variational recurrent auto-encoders and generative adversarial networks. Our main hypothesis is that the learned representation of the summary video and the original video should be similar. The summarizer aims to summarize the video such that the discriminator is fooled and the discriminator aims to recognize the summary videos from original videos. The entire model is trained in an adversarial manner where the GAN’s discriminator is used to learn a similarity measure for training the recurrent encoder/decoder and the frame selector LSTMs. Variations of our approach are defined using different regularizer terms. Evaluation on benchmark datasets demonstrate that all unsupervised variations of our approach outperform the state of the art in video summarization and provides accuracy comparable to state-of-the-art supervised approaches.

Fourth, we presented a generative framework using GANs for human 3D pose estimation where predicting specific joint locations is fine-grained recognition of joints as parts from images. We proposed a novel generator network that uses part-based modeling with LSTMs. An ablation study was performed to determine the best generator architecture. We verified our hypothesis that GAN training does yield a smooth latent space of 3D poses. Overall, we demonstrated that our GAN architecture is effective for human 3D pose estimation.

Fifth, we presented an alternate generative framework for human 3D pose estimation, where again predicting specific joint locations is fine-grained recognition of joints as parts from images. This GAN framework not only outperforms the state of the art but also provides explanations of predictions. Our key contribution is a new way to disentangle factors of variation in a GAN and thus enable interpretability of its latent feature space, which in turn can be used to provide explanations of predictions. The disentanglement is formalized by parameterizing the latent space of the GAN as a Gaussian mixture model. The Gaussian mixture is regularized such that its components represent canonical human poses seen in training data. We also introduced a new progressive training strategy that stabilizes GAN training and allows learning of human 3D poses to much higher accuracy. Finally, we demonstrated that adding the new interpretability objective in our GAN leads to performance improvements over the standard GANs and superior results relative to the state of the art. We showed that we are able to disentangle the factors of variation in the data and therefore addressed our hypothesis.

Finally, in all chapters, we demonstrated the validity of our hypotheses and evaluated our contributions with respect to baselines and corresponding state-of-the-art approaches. We demonstrated that our contributions lead to superior results relative to the state of the art.
Bibliography


