2016

Deep Learning for Decision Making and Autonomous Complex Systems

Kin Gwn Lore

Iowa State University

Follow this and additional works at: http://lib.dr.iastate.edu/etd

Part of the Computer Sciences Commons, and the Mechanical Engineering Commons

Recommended Citation
http://lib.dr.iastate.edu/etd/15965

This Thesis is brought to you for free and open access by the Iowa State University Capstones, Theses and Dissertations at Iowa State University Digital Repository. It has been accepted for inclusion in Graduate Theses and Dissertations by an authorized administrator of Iowa State University Digital Repository. For more information, please contact digirep@iastate.edu.
Deep learning for decision making and autonomous complex systems

by

Kin Gwn Lore

A thesis submitted to the graduate faculty
in partial fulfillment of the requirements for the degree of
MASTER OF SCIENCE

Major: Mechanical Engineering

Program of Study Committee:
Soumik Sarkar, Major Professor
Baskar Ganapathysubramanian
Chinmay Hegde

Iowa State University
Ames, Iowa
2016

Copyright © Kin Gwn Lore, 2016. All rights reserved.
DEDICATION

I would like to dedicate this thesis to my major professor, Dr. Soumik Sarkar. Without his constant support and expert guidance, I would never have been able to complete this work with enthusiasm and become more well-versed in this field.

I would also like to dedicate this thesis to my family for their unconditional love and persistent support when completing this work.
TABLE OF CONTENTS

LIST OF TABLES viii

LIST OF FIGURES x

ACKNOWLEDGEMENTS xxii

ABSTRACT xxiii

CHAPTER 1. INTRODUCTION 1

CHAPTER 2. DEEP LEARNING PRELIMINARIES 5

2.1 Architectures ........................................ 5
  2.1.1 Restricted Boltzmann machines ..................... 5
  2.1.2 Deep neural networks ............................. 9
  2.1.3 Convolutional neural networks ................... 12
  2.1.4 Stacked autoencoders ........................... 16

2.2 Regularization ...................................... 18
  2.2.1 Weight decay .................................. 19
  2.2.2 Early stopping ................................ 19

2.3 Feature visualization ............................... 20
PART I  DEEP LEARNING FOR DECISION MAKING IN MECHANICAL SYSTEMS

CHAPTER 3. DATA-DRIVEN DEEP LEARNING MODELS FOR EFFICIENT DESIGN OF MICROFLUIDIC FLOW PATTERNS

3.1 Motivation .................................................. 24
3.2 Problem setup ............................................. 27
3.3 Related approach: Genetic Algorithm ...................... 28
3.4 Sequence prediction by Simultaneous Multiple Classification ............ 28
  3.4.1 Training data generation using a metric on the output space .......... 30
  3.4.2 Experiments and results .................................. 32
3.5 Action sequence learning for causal shape transformation ............. 37
  3.5.1 Proposed architectures .......................... 39
  3.5.2 Results and discussions ......................... 44
3.6 Conclusions and Future Work ................................ 46

CHAPTER 4. EARLY DETECTION OF COMBUSTION INSTABILITY USING HI-SPEED FLAME VIDEOS

4.1 Motivation .................................................. 49
4.2 Experimental Setup ......................................... 52
4.3 Deep learning and symbolic time series analysis: an integrated framework ...... 55
4.4 Background on symbolic time series analysis (STSA) ................ 55
  4.4.1 Generalized D-Markov machine [123] ...................... 57
  4.4.2 Construction of a D-Markov machine [94] .............. 58
  4.4.3 Cross modeling for sensor fusion ........................ 60
4.5 Early detection with DBN+STSA .............................. 62
  4.5.1 DBN feature visualization ............................ 63
  4.5.2 Performance of STSA module .................. 65
PART II  DEEP LEARNING FOR AUTONOMOUS COMPLEX SYSTEMS  

CHAPTER 5.  DEEP VALUE OF INFORMATION ESTIMATORS FOR COLLABORATIVE HUMAN-MACHINE INFORMATION GATHERING  

5.1  Motivation  

5.2  Human-machine collaboration  
5.2.1  Problem setup  
5.2.2  Value of information (VOI) for soft/hard sensor scheduling  

5.3  Deep VOI estimators  
5.3.1  Design of framework  

5.4  Results and discussion  
5.4.1  Prediction accuracy  
5.4.2  Comparison with AMDP  

5.5  Semantic soft data scheduling  

5.6  Conclusions and future work  

4.6  Early detection with CNN+STSA  
4.6.1  CNN training  
4.6.2  STSA-based instability measure with CNN  
4.6.3  Comparison of CNN+STSA with PCA+STSA  

4.7  Early detection with multimodal sensor fusion  

4.8  Early detection using deep convolutional selective autoencoders  
4.8.1  Convolutional Selective Autoencoder  
4.8.2  Instability measure  
4.8.3  Implementation  
4.8.4  Performance Evaluation  

4.9  Conclusion and Future Work  

5.1  Motivation  

5.2  Human-machine collaboration  
5.2.1  Problem setup  
5.2.2  Value of information (VOI) for soft/hard sensor scheduling  

5.3  Deep VOI estimators  
5.3.1  Design of framework  

5.4  Results and discussion  
5.4.1  Prediction accuracy  
5.4.2  Comparison with AMDP  

5.5  Semantic soft data scheduling  

5.6  Conclusions and future work  

v
CHAPTER 6. LLNET: A DEEP AUTOENCODER APPROACH TO NATURAL LOW-LIGHT IMAGE ENHANCEMENT

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.1</td>
<td>Motivation</td>
<td>109</td>
</tr>
<tr>
<td>6.2</td>
<td>Related work</td>
<td>111</td>
</tr>
<tr>
<td>6.3</td>
<td>The Low-light Net (LLNet)</td>
<td>112</td>
</tr>
<tr>
<td>6.3.1</td>
<td>Learning features from low-light images with LLNet</td>
<td>112</td>
</tr>
<tr>
<td>6.3.2</td>
<td>Network parameters</td>
<td>113</td>
</tr>
<tr>
<td>6.3.3</td>
<td>Training data generation</td>
<td>114</td>
</tr>
<tr>
<td>6.3.4</td>
<td>Simulating darkness</td>
<td>114</td>
</tr>
<tr>
<td>6.3.5</td>
<td>Image reconstruction</td>
<td>116</td>
</tr>
<tr>
<td>6.4</td>
<td>Evaluation metrics and compared methods</td>
<td>116</td>
</tr>
<tr>
<td>6.4.1</td>
<td>Performance metric</td>
<td>116</td>
</tr>
<tr>
<td>6.4.2</td>
<td>Compared methods</td>
<td>118</td>
</tr>
<tr>
<td>6.5</td>
<td>Results and discussion</td>
<td>120</td>
</tr>
<tr>
<td>6.5.1</td>
<td>Algorithm adaptivity</td>
<td>120</td>
</tr>
<tr>
<td>6.5.2</td>
<td>Enhancing artificially darkened images</td>
<td>122</td>
</tr>
<tr>
<td>6.5.3</td>
<td>Enhancing darkened images in the presence of synthetic noise</td>
<td>123</td>
</tr>
<tr>
<td>6.5.4</td>
<td>Application on natural low-light images</td>
<td>126</td>
</tr>
<tr>
<td>6.5.5</td>
<td>Training with Gaussian vs. Poisson noise</td>
<td>127</td>
</tr>
<tr>
<td>6.5.6</td>
<td>Denoising capability, image sharpness, and patch size</td>
<td>129</td>
</tr>
<tr>
<td>6.5.7</td>
<td>Prior knowledge on input</td>
<td>130</td>
</tr>
<tr>
<td>6.5.8</td>
<td>Features of low-light images</td>
<td>132</td>
</tr>
<tr>
<td>6.5.9</td>
<td>Hyper-parameters, network architecture, and performance</td>
<td>134</td>
</tr>
<tr>
<td>6.5.10</td>
<td>Color implementation</td>
<td>134</td>
</tr>
<tr>
<td>6.6</td>
<td>Conclusions and future works</td>
<td>135</td>
</tr>
</tbody>
</table>
CHAPTER 7. ROOT-CAUSE ANALYSIS FOR TIME-SERIES ANOMALIES VIA SPATIOTEMPORAL CAUSAL GRAPHICAL MODELING

7.1 Motivation ................................................................. 138
7.2 Background and preliminaries ................................. 140
  7.2.1 Unsupervised anomaly detection with spatiotemporal graphical modeling 140
7.3 Methods ................................................................. 141
  7.3.1 Sequential state switching ($S^3$) ......................... 141
  7.3.2 Artificial anomaly association ($A^3$) ............... 144
7.4 Results and discussions ........................................ 145
  7.4.1 Anomaly in pattern(s) ....................................... 145
  7.4.2 Anomaly in node(s) ........................................ 147
  7.4.3 Discussions .................................................. 151
7.5 Conclusions ......................................................... 151

CHAPTER 8. SUMMARY AND CONCLUSION ........................... 153
LIST OF TABLES

Table 3.1 Number of possible sequences, $n_s$ in an $n_p$-pillar sequence. ........................................ 31
Table 3.2 PMR-based performance for DNN vs CNN. The appended number 4 is the number of pillars in the sequence. .......................................................... 33
Table 3.3 PMR-based performance for CNN with 7-pillar sequence that is trained with randomly-generated training data vs. quasi-randomly generated training data. .......................................................... 33
Table 3.4 PMR and SSIM of regenerated flow shapes using different enhancement methods. The numbers reported are in the format of [PMR/SSIM]. Bolded numbers correspond to the method with the highest PMR and SSIM. Asterisk (*) denotes our architecture presented in this paper. .................. 47
Table 4.1 Description of operating conditions along with respective ground truth (stable or unstable) for hi-speed image data collection. 3s of greyscale image sequence at 3kHz is collected for each condition . . . . . . . . . . . . . 54
Table 6.1 PSNR and SSIM of outputs using different enhancement methods. ‘Bird’ means the non-dark and noiseless (i.e. original) image of Bird. ‘Bird-D’ indicates a darkened version of the same image. ‘Bird-D+GN18’ denotes a darkened Bird image with added Gaussian noise of $\sigma = 18$, whereas ‘Bird-D+GN25’ denotes darkened Bird image with added Gaussian noise of $\sigma = 25$. Bolded numbers corresponds to the method with the highest PSNR or SSIM. Asterisk (*) denotes our framework. .................. 121
Table 6.2  Average PSNR and SSIM over 90 synthetic and 6 natural test images. Synthetic test images are randomly darkened with $\gamma \in [1, 4]$ and Gaussian noise levels of $\sigma \in [0, 25]$. Natural test images are taken under natural low-light conditions. Because gamma darkening is performed randomly for this set of images, we search for the optimal $\gamma$ parameter that results in the highest SSIM ($\gamma = 0.05 : 0.05 : 1$) when applying gamma adjustment. Note that searching for the optimal parameter is infeasible in reality because no reference image is available. The number reported within the parentheses is the number of winning instances among 90 synthetic test images and 6 natural test images. Asterisk (*) denotes our framework.

Table 6.3  Average PSNR evaluated on the set of 90 test images enhanced with trained model of different hyper-parameters and network architecture. The implemented model is marked by an asterisk (*), whereas the PSNR and SSIM for the best model are presented in bolded typeface.

Table 7.1  Root-cause analysis results in $S^3$ method and $A^3$ method with synthetic data.

Table 7.2  Comparison of root-cause analysis results with $S^3$ and VAR.
LIST OF FIGURES

Figure 2.1  Distinction between Boltzmann machines and restricted Boltzmann machines. In RBMs, there are no interconnectivity between units within the same layer (i.e. visible-visible or hidden-hidden connections). 6

Figure 2.2  Simple and complex features for faces, cars, elephants, chairs, and other objects. Figure courtesy of Lee et al. [74]. 10

Figure 2.3  Stacking restricted Boltzmann machines, RBMs (a) to form deep belief networks, DBN (b). Adding a target layer on top will yield a deep neural network, DNN (c). 10

Figure 2.4  Structure of a CNN with multiple layers of convolutional and pooling layers before the fully connected hidden layers. 13

Figure 2.5  The use of shared weights reduces memory footprints, preserves local correlations, and reduces the number of learnable parameters to prevent overfitting. The colors of the sparse connections denote shared weights. 14

Figure 2.6  Convolving a filter (orange) over the input image (blue) to generate a convolved feature map (green). For each output, the values of the filters are multiplied element-wise with the overlapping region in the input image. The products are summed up as one output element in the convolved feature. 14

Figure 2.7  The maxpooling scheme. The feature maps (blue) are divided into non-overlapping partitions (shaded blues) where the maximum value is selected from each partition to form a new matrix (green). 15
Figure 2.8 Learning the identity function with an autoencoder (i-input, h-hidden unit, o-output). (a) The hidden layer is an identity function where the input can be fully reconstructed without loss. (b) Reduced number of hidden units captures the most important features to effectively reconstruct the input. (c) Sparsity constraints suppress activations of the hidden units to help extracting low-dimensional features from the input. .......................... 16

Figure 2.9 Training a single layer of denoising autoencoder. Input vector $x$ is corrupted to obtain $\tilde{x}$. The objective is to learn the parameter $\theta$ that is able to reconstruct $z$ from a compressed representation of $\tilde{x}$ such that the error between $z$ and $x$ is minimized. The same parameter $\theta$ is used in both the encoding and the decoding phase. .......................... 17

Figure 2.10 Stacking multiple autoencoders to form a deep architecture. The layers are trained separately. .......................... 18

Figure 2.11 Effects of varying regularization parameter. Panel (a) shows a perfect network being the ideal case. Panels (b) to (e) shows what happens when the regularization parameter $\lambda$ is increased. Panel (f) shows the situation where $\lambda = \infty$. .......................... 19

Figure 2.12 The early stopping algorithm prevents overfitting by evaluating losses on both the training set and the validation set. .......................... 20

Figure 3.1 Pillar programming. Each pillar contributes to the deformation of the flow. The position-diameter pair of each pillar is assigned an index which will be used as class labels for classification. .......................... 27

Figure 3.2 CNN with the SMC problem formulation. .......................... 29

Figure 3.3 Uniform sampling vs. Quasi-random sampling. A set of 160 2-pillar sequences are formed using uniform (left) and Sobol (right) sampling methods. The first 10 sampled sequences are shown in red, the next 50 sampled sequences in green, and the final 100 sampled sequences in blue. .......................... 31
Figure 3.4 Eight example predictions ($n_p = 4$). The left side of each pair is the target flow shape and the right side is the flow generated from the predicted pillar sequences.

Figure 3.5 Visualization of hidden layer activations in the DNN.

Figure 3.6 Visualization of CNN filters and feature maps of a test sample from convolutional layers 1 and 2.

Figure 3.7 Comparison of pixel match rate and algorithm runtime for GA versus DL methods (using CNN-SMC) to generate pillar sequences that has the best shape reconstruction closest to the target flow shape. Four examples are provided.

Figure 3.8 (a) Variation of the average PMR and training size ratio (ratio of training samples to the number of all possible pillar sequences, on a logarithmic scale) with increasing number of pillars in a sequence. (b) Flow shapes generated from predicted pillar sequences.

Figure 3.9 The PPN addresses the question: ‘Given a pair of pre- and post-deformed shape, what is the identity of the pillar causing the deformation?’

Figure 3.10 PPN-C treats the pre- and post-deformed shapes as separate channels.

Figure 3.11 The ITN addresses the question: ‘Given the final shape, what is the possible shape that lies in the middle of the nonlinear deformation pathway?’

Figure 3.12 The integrated pipeline combining both PPN and ITN.

Figure 3.13 Four examples of sequence prediction using (i) PPN-only without bridging and (ii) PPN+ITN with bridging. By predicting a bridging shape, the resulting predicted sequence is able to reconstruct flow shapes that are more similar to the target shape. Each frame shows the deformation on the flow shape with each additional predicted pillar added into the sequence.

Figure 3.14 Sample-wise (a) PMR and (b) SSIM comparison using CNN+SMC, PPN, PPN-C, and PPN+ITN. 20 test target shapes are randomly generated with a 10-pillar sequence.
Figure 3.15 20 test shapes with reconstructed flow shapes generated from sequences predicted using different methods. Column A is the target shape, B for the reconstruction for CNN+SMC, C for PPN, D for PPN-C, and E for PPN+ITN.

Figure 4.1 (a) Schematic of the experimental setup. 1 - settling chamber, 2 - inlet duct, 3 - IOAM, 4 - test section, 5 - big extension duct, 6 - small extension ducts, 7 - pressure transducers, $X_s$ - swirler location measured downstream from settling chamber exit, $X_p$ - transducer port location measured downstream from settling chamber exit, $X_i$ - fuel injection location measured upstream from swirler exit, (b) Swirler assembly used in the combustor.

Figure 4.2 Top: greyscale images at $Re = 7,971$ and full premixing for a fuel flow rate of 0.495 g/s, bottom: greyscale images at $Re = 15,942$ and full premixing for a fuel flow rate of 0.495 g/s.

Figure 4.3 Framework for early detection of combustion instability from hi-speed flame images via semantic feature extraction using deep belief network (DBN) followed by symbolic time series analysis (STSA).

Figure 4.4 Variation of $\times D$-Markov entropy rate directed from time-series of DBN hidden units (output of hi-speed video) to pressure time series as a function of number of state splitting at a symbol size $|\Sigma| = 3$ for (a) stable combustion, (b) thermo-acoustically unstable combustion.

Figure 4.5 (d) Visualization of weights from the first layer and inputs that maximizes the hidden unit activations for the (c) 1st layer, (b) 2nd layer, and (a) 3rd layer after pre-training and prior to supervised finetuning.

Figure 4.6 (d) Visualization of weights from 1st layer and inputs that maximizes the hidden unit activations for the (c) 1st layer, (b) 2nd layer, and (a) 3rd layer after supervised finetuning.
Figure 4.7 0.2s long time series of $l_2$ norms of (i) 10 largest variance components of PCA performed on images at (a) stable and (b) unstable states and (ii) activation probabilities of last hidden layer after pre-training a DBN on images at (c) stable and (d) unstable states. 65

Figure 4.8 Variation of Euclidean distance between STSA features of image sequences from stable and unstable combustion as a function of alphabet size for STSA. 66

Figure 4.9 Filter visualization at convolutional layer (a) one and (b) two. (b) shows fragmented representations of coherent structures that are visible in unstable flame. (c) Feature maps of a stable frame (top) and an unstable frame (bottom) after applying first convolutional layer filter. Red outline on the unstable flame visualization shows how the mushroom-shaped coherent structure is highlighted. 68

Figure 4.10 Variation of the proposed instability measure with time for the transition video named 600_{50to35}. Multiple regions on the measure curve denote different combustion states such as stable, temporary intermittancy (a significant precursor to persistent instability) and unstable. They are corresponded to varied coherent structures (bounded by red box) that are detected by the ‘CNN+STSA’ framework. On the right, $\text{rms}$ variation of the pressure is shown as it is one of the most commonly used instability measures. Progression of $P_{\text{rms}}$ can not detect the aforementioned precursors. 69

Figure 4.11 (a) Comparison of sudden change in instability measure when instability sets in for different transition conditions which are 1. 500_{40to38}, 2. 500_{40to30}, 3. 40_{500to600}, 4. 600_{50to35} and 5. 50_{700to800}. The jump is larger for ‘CNN+STSA’ than ‘PCA+STSA’. (b) Variation of instability measure for both ‘CNN+STSA’ than ‘PCA+STSA’ at transition condition 50_{700to800}. The measure arising from ‘CNN+STSA’ can detect the intermittancy precursor whereas it is mostly ignored by ‘PCA+STSA’. A frame with an intermittancy coherent structure in a red box is shown on the top. 70
Figure 4.12 Entropy rate as an instability measure at stable, relatively stable and unstable states for (a) D-Markov machine on pressure time series, (b) $\times D$-Markov machine directed from hi-speed video dominant-PCA-feature-aggregate time series to pressure time series, (c) $\times D$-Markov machine directed from hi-speed video bottleneck layer 5th hidden unit to pressure time series and (d) 10th hidden unit to pressure time series . ... 73

Figure 4.13 Grayscale images of gradual time-varying development of instability structure at two different parameter values . ... 75

Figure 4.14 Structure of the convolutional autoencoder with associated layer parameters. The encoder portion extracts meaningful features from convolution and sub-sampling operations, while the decoder portion reconstructs the output into the original dimensions through deconvolution and upsampling. Unpooling is done with $2 \times 2$ upsampling by replicating the elements. Best viewed on screen, in color. ... 77

Figure 4.15 Schematics of implementation of trained network on transition test data . 79

Figure 4.16 Illustration of the pipeline’s ability to reproduce explicit labels. ... 80

Figure 4.17 Feature maps for (a) the third convolution layer, (b) the second pooling layer, (c) the fourth convolution layer, (d) the unpooling layer and (e) the deconvolution layer. ... 81

Figure 4.18 Results of transition protocols for: a) $600_{50 \to 35}$, b) $500_{40 \to 30}$ and c) $50_{700 \to 800}$ where purple arrows indicate expected results from stability and instability while the green arrows with different intensities indicate the strength of early instability presence in a supposedly stable region. ... 82

Figure 4.19 Adjacency labeling result of transition protocols for: a) $600_{50 \to 35}$, b) $500_{40 \to 30}$ and c) $50_{700 \to 800}$ with three consecutive frames on a transition protocol highlighted ... 83
Figure 5.1  Sample truth model simulation of 2D target localization problem, showing
locations of cameras 1-6 and associated fields of views, along with posterior
distribution $p(X_k|O_{1:k})$ (heat map) and true target location (magenta
x) for a random walk motion model $p(X_{k+1}|X_k)$. Each sensor has 0.99
detection rate and false alarm rate of $1 \times 10^{-4}$, 0.1504, 0.1585, 0.1992,
0.1510, and 0.1593, respectively. ........................................... 92

Figure 5.2  Schematic of the CNN used for reward learning. ......................... 97

Figure 5.3  Four examples of belief map and VOI (discretized into bins) associated
with the action of switching into a particular sensor at different simulation
time steps $T$. ................................................................. 99

Figure 5.4  Estimated VOI vs. True VOI for the first 40 out of 10,000 simulation
time steps. The figure also shows whether the sensor selected through
rank-ordering the predicted VOI is the same as the sensor selected by
ordering the true VOI, represented by green circles (matching) and red
crosses (non-matching). .................................................. 99

Figure 5.5  Degradation of selection accuracy with reduction of dataset sizes used to
train the CNN model. .......................................................... 100

Figure 5.6  Entropy of belief map over time. .......................................... 102

Figure 5.7  Euclidean error between MAP estimate and target location over time. .. 103

Figure 5.8  Error distribution between MAP estimate and target location. ........ 103

Figure 5.9  The indoor search environment and associated semantic features for soft
observations. ................................................................. 105

Figure 5.10 The likelihood function that maps to the example observation, ‘Target is
near the dining table’. ....................................................... 106

Figure 5.11 Nine examples of CNN input showing the belief map, padding, and the
action map in the realistic scenario. ....................................... 107
Figure 6.1 Architecture of the proposed framework: (a) An autoencoder module is comprised of multiple layers of hidden units, where the encoder is trained by unsupervised learning, the decoder weights are transposed from the encoder and subsequently fine-tuned by error back-propagation; (b) LLNet with a simultaneous contrast-enhancement and denoising module; (c) S-LLNet with sequential contrast-enhancement and denoising modules. The purpose of denoising is to remove noise artifacts often accompanying contrast enhancement. .......................................................... 113

Figure 6.2 Training the LLNet: Training images are synthetically darkened and added with noise. These images are fed through LLNet where the reconstructed images are compared with the uncorrupted images to compute the error, which is then back-propagated to finetune and optimize the model weights and biases. .......................................................... 117

Figure 6.3 Original standard test images used to compute PSNR. ................. 122

Figure 6.4 Comparison of methods of enhancing ‘Town’ when applied to (A) original already-bright, (B) darkened, (C) darkened and noisy ($\sigma = 18$), and (D) darkened and noisy ($\sigma = 25$) images. Darkening is done with $\gamma = 3$. The numbers with units dB are PSNR, the numbers without are SSIM. Best viewed on screen. .......................................................... 123

Figure 6.5 Comparison of methods on randomly darkened/noise-added synthetic test images of (A) Bird, (B) Girl, (C) Pepper, and (D) House. Darkening and noise addition are done using randomized values of $\gamma \in [1, 4]$ and $\sigma \in [0, 25]$. The numbers with units dB are PSNR, the numbers without are SSIM. Best viewed on screen. ......................... 124
Figure 6.6 Comparison of methods of enhancing naturally dark images of (A) chalkboard, (B) computer, (C) objects, (D) chart, (E) cabinet, and (F) writings. Selected regions are enlarged to demonstrate the denoising and local contrast enhancement capabilities of LLNet. HE (including HE+BM3D) results in overamplification of the light from the computer display whereas LLNet was able to avoid this issue. ................................. 125

Figure 6.7 The result of adding Gaussian noise and applying Poisson noise with increasing photon count (normalized from 0 to 1 with a step size of 0.1). Although the noise levels between the two noise types look similar at higher photon count, the first three columns look very different. Best viewed on screen. ................................................................. 127

Figure 6.8 Natural low-light image enhancement results of LLNet trained with Gaussian noise (LLNet-G) and Poisson noise (LLNet-P). ‘Gsn’ and ‘Psn’ are abbreviations for Gaussian and Poisson respectively. Best viewed on screen. 128

Figure 6.9 Relative patch size vs PSNR and SSIM. The picture with highest PSNR has the highest denoising capability but least sharp. Picture with lowest $r$ has the least denoising capability but has the highest image sharpness. Picture with the highest SSIM balances between image sharpness and denoising capability. ................................. 130

Figure 6.10 Evaluation on US Air Force (USAF) resolution test chart. There exist optimal relative patch sizes that result in the highest PSNR or SSIM after image enhancement (using LLNet). Note that the result enhanced with histogram equalization is shown to highlight the loss in detail of the natural dark image (where the main light is turned off) compared to the natural bright image. ................................. 131

Figure 6.11 Feature detectors can be visualized by plotting the weights connecting the input to the hidden units in the first layer. These weights are selected randomly. ................................. 132
Figure 6.12 Random selection of weights from the first layer (feature detectors) and
weights from the output layer (feature generators) for the integrated LLNet
model trained with a patch size of $21 \times 21$. Patterns in the output weights
are similar to patterns in the first hidden layer weights since tied weights
are used. 132

Figure 6.13 Random selection of first layer weights from an integrated LLNet model
trained in batches of 1000 and 50, respectively. The superior model (i.e.
batch size 50) learns features that appear more distinctive. 133

Figure 6.14 Enhancing natural low-light colored image with alternative methods. From
left to right: Natural dark, histogram equalization, gamma adjustment,
histogram equalization with Color-BM3D (CBM3D), and LLNet. Best
viewed on screen, in color (full resolution available when zoomed in). 137

Figure 7.1 Anomaly detection process. Spatiotemporal features are extracted from
both nominal and anomalous data, with online STPN applied. Multiple
sub-sequences of APs and RPs form input vectors to the RBM. Here, the
RBM is only trained with nominal data, and the anomalous data is used
as input to compute the free energy. Anomaly is detected by identifying
its high energy state, which can be captured using the Kullback-Leibler
Distance (KLD) metric. 141

Figure 7.2 Framing the problem as an artificial anomaly association ($A^3$) problem.
(a) When training the model, the value of an element in both the input
vector and its corresponding indicator label is randomly flipped to simulate
anomaly. (b) When inferencing, a test input is fed into the DNN and a
classification sub-problem is solved to obtain the indicator vector. Values
of 0 in the output vector traces back to the exact patterns that are faulty. 145
Figure 7.3 Graphical models defined to simulate pattern(s) anomaly. Six graphs are defined and treated as nominal operation modes in complex systems. Pattern failure is simulated by breaking specific patterns in the model (not shown).

Figure 7.4 Anomalous conditions with fault node, causality is discovered by VAR model. Compared with the nominal mode in Fig. 7.3 (a), the fault node breaks most of the causality from and to this node. Causality is normalized by the maximal value among all of the patterns, and causality smaller than 0.03 is not shown.

Figure 7.5 Comparisons between sequential state switching ($S^3$) method and vector autoregressive (VAR) method using dataset3. The patterns represented with blocks are from the node in x-axis to the node in y-axis, and the discovered root causes are marked in black.
ACKNOWLEDGEMENTS

Continuing my graduate education has been a life-changing, unforgettable, and absolutely enjoyable journey of my life. I would like to use the opportunity to express my gratitude to everyone who has inspired me and helped me with various aspects in conducting the research.

First and foremost, I would like to express my sincerest gratitude and appreciation to Dr. Soumik Sarkar for his immeasurable amount of guidance, patience and support throughout this research and the writing of this thesis. His insights and words of encouragement are truly a blessing to me and have often inspired me to become an excellent, well-rounded practitioner of science. He is certainly a great mentor, a thoughtful friend, and a role model. I would also like to thank my committee members, Dr. Baskar Ganapathysubramanian and Dr. Chinmay Hegde, for providing insightful advice, efforts and contributions to this work.

I am especially grateful to my family for constantly offering emotional support while enrolled in the graduate program. I would also like to thank my friends in the laboratory and student office who have been best friends to share the joys of successful achievements and the griefs of having papers rejected for publication. Thank you all for being there for me during both good and bad times, and for making my time as a graduate student extremely delightful and memorable.

I would like to thank all of my collaborators. Without their invaluable opinions and expertise, I would never have produced tangible results from my research efforts. Along with my collaborators and co-authors, we sincerely acknowledge the extensive data collection performed by Vikram Ramanan and Dr. Satyanarayanan Chakravarthy at Indian Institute of Technology Madras (IITM), Chennai. The work has been supported in part by the National Science Foundation under Grant No. CNS-1464279, Iowa State Regents Innovation Funding, and Rockwell Collins Inc. We also gratefully acknowledge the support of NVIDIA Corporation with the donation of the GeForce GTX TITAN Black GPU used for this research.
ABSTRACT

Deep learning consists of various machine learning algorithms that aim to learn multiple levels of abstraction from data in a hierarchical manner. It is a tool to construct models using the data that mimics a real world process without an exceedingly tedious modelling of the actual process. We show that deep learning is a viable solution to decision making in mechanical engineering problems and complex physical systems.

In this work, we demonstrated the application of this data-driven method in the design of microfluidic devices to serve as a map between the user-defined cross-sectional shape of the flow and the corresponding arrangement of micropillars in the flow channel that contributed to the flow deformation. We also present how deep learning can be used in the early detection of combustion instability for prognostics and health monitoring of a combustion engine, such that appropriate measures can be taken to prevent detrimental effects as a result of unstable combustion.

One of the applications in complex systems concerns robotic path planning via the systematic learning of policies and associated rewards. In this context, a deep architecture is implemented to infer the expected value of information gained by performing an action based on the states of the environment. We also applied deep learning-based methods to enhance natural low-light images in the context of a surveillance framework and autonomous robots. Further, we looked at how machine learning methods can be used to perform root-cause analysis in cyber-physical systems subjected to a wide variety of operation anomalies. In all studies, the proposed frameworks have been shown to demonstrate promising feasibility and provided credible results for large-scale implementation in the industry.
CHAPTER 1. INTRODUCTION

Deep learning is a set of machine learning algorithms that aim to learn multiple levels of abstraction from data in a hierarchical manner. In other words, such algorithms operate on high-dimensional data that arrives in high volume, velocity, variety, veracity, and value. Oftentimes, solutions to various physical problems often involve constructing rigorous physical models to explain the observed phenomena in a highly detailed manner without neglecting the various sources of disturbance from the environment. Traditionally, engineers can only devise feasible solutions to a problem after laborious and time-consuming analysis, modeling, and data interpretation in producing reliable models. Due to the rapid advancement of technology, more and more problems need to be solved; there is a growing necessity to solve such problems as quickly and accurately as current technology allows. Unfortunately, it is an immensely tedious endeavour to construct new models all the time, which may potentially become too narrow in scope and limited in extensibility to other domains. Under these constraints, a scalable and fast modeling tool is desired.

Many domains such as engineering involve extensive collection and maintenance of large volumes of raw data for analysis and interpretation. As a data-driven approach, deep learning becomes an increasingly attractive option to create models based on these data generated by complex physical processes and simulation. A huge advantage is that the available data has already accounted for all disturbances and uncertainties coming from the natural environment. Further, deep learning algorithms perform feature extraction in an automated manner without feature hand-crafting; this powerful aspect of deep learning can be leveraged for high-level decision making. We show that in many cases, features extracted by deep learning algorithms from the data can provide sufficient insights to solve various problems without the overly-complex modeling of a physical phenomenon.
Deep learning is widely used in many problem areas ranging from object recognition [139, 40] and speech recognition [47, 29] to robotic perception [65] and human disease prediction [82]. In our study reported in Chapter 3 [84, 85], we demonstrate a novel application of deep learning in a mechanical design problem. Specifically, we learn from complex microfluidic flow patterns in order to solve inverse problems in fluid mechanics. A recent discovery enabled engineers to control the deformation of the cross-sectional shape of a flow by deliberately placing a sequence of pillars with varying positions and diameters [7]. This groundbreaking effort opens the door to rapid advancement in material science, manufacturing and biological applications. However, designing the sequences of pillars for user-defined deformations requires laborious and time-consuming design iterations. The problem is also exacerbated when the nonlinear design space has as large as \(10^{15}\) possibilities of flow shapes for a 10-pillar sequence. We demonstrate that hierarchical feature extraction techniques lead to scalable design tools by learning semantic representations from a relatively small number of flow patterns examples. In this study, we proposed various architectures including deep neural networks (DNN), convolutional neural networks (CNN), and stacked autoencoders (SAE) and compared their performance to the current state-of-the-art genetic algorithm (GA). Results show that the deep learning based design process is shown to expedite the current state-of-the-art design approach by over 600 times with competitive performance.

We also worked with our collaborators (as co-authors) to design a framework in the context of prognostics and health monitoring with relevant results published in [122, 121, 6, 117] and shown in Chapter 4. Specifically, we propose a neural-symbolic framework for analyzing a large volume of sequential high-speed images of flames in engines for the early detection of combustion instability that is extremely critical for engine health. The proposed hierarchical approach involves extracting low-dimensional semantic features from images using deep CNNs followed by capturing the temporal evolution of the extracted features using symbolic time series analysis (STSA). The semantic nature of the CNN features enables expert-guided data exploration that can lead to better understanding of the underlying physics. Extensive experimental data have been collected in a swirl-stabilized dump combustor at various operating conditions for validation.
Another application concerning the robotic path planning via effective human-machine collaboration is published as [86] and presented in Chapter 5. An effective collaboration may improve a large number of learning and planning strategies to gather information through the fusion of hard data originated from machine sensors and soft data originated from human sensors. However, gathering the most informative data from the human counterparts without task overloading remains a technical challenge. In this context, Value of Information (VOI) is a crucial decision-theoretic metric for scheduling interaction with human sensors. Naturally, deep learning becomes an attractive method to be used as a VOI estimation framework to schedule collaborative human-machine sensing with the advantage of computationally efficient online inference and minimal policy hand-tuning. We propose that supervised feature extraction from the image representation of the belief space using CNNs can be associated with soft data query choices to compute the corresponding VOI for human interaction in a reliable manner. The performance of the proposed framework is compared with a feature-based scheduling policy modeled using the partially-observed Markov decision process (POMDP). The practical feasibility of the method is demonstrated on a realistic mobile robotic search problem with language-based semantic human sensor inputs.

Mobile robots are also frequently deployed as part of a surveillance and monitoring framework as well as in tactical reconnaissance. In this context, gathering visual information from a dynamic environment and accurately processing such data are essential to making informed decisions and ensuring the success of a mission. Cameras are often required to capture images or videos in a lowly illuminated environment. Therefore, it is desirable to perform brightness and contrast enhancement and simultaneously reduce noise content from the image frames in an on-board real-time manner. In Chapter 6, we tackle the problem of natural low-light image enhancement using deep autoencoders where the proposed algorithm is able to adaptively enhance image contrast without saturation effects while reducing image noise as a result of contrast enhancement. We show that a variant of the stacked-sparse denoising autoencoder can learn from synthetically darkened and noise-added training examples to enhance images taken from natural low-light images. The chapter presents the published article from [83] with additional results.
Aside from human-machine interaction systems, cyber-physical systems play a huge role in many engineering processes. It is not uncommon for modern distributed cyber-physical systems to encounter anomalies in any cases. When neglected, these systems are vulnerable to catastrophic fault propagation due to the strong connectivity between different sub-systems. Analyzing the root cause may become highly intractable due to the complex mechanism of fault propagation with a large variety of operating modes. In Chapter 7 (published as [80]), we present a data-driven framework using machine learning methods to circumvent these issues in root-cause analysis. We propose the technique of sequential state switching ($S^3$) using restricted Boltzmann machines (RBM) and artificial anomaly association ($A^3$) using DNN for root-cause analysis in an unsupervised and semi-supervised manner respectively. Synthetic data from cases with failed patterns and anomalous node are simulated to validate the proposed approaches, then compared with the performance of vector autoregressive (VAR) model-based root-cause analysis. We show that these machine learning based methods result in satisfactory performance in terms of accuracy and capability in handling multiple modes.
CHAPTER 2. DEEP LEARNING PRELIMINARIES

In this chapter, a brief background on key deep learning concepts as well as the various architectures used in our studies is presented.

2.1 Architectures

There are various deep architectures available at our disposal to solve a variety of different problems. Here, we will present the theory behind restricted Boltzmann machines, deep belief networks, deep neural networks, convolutional neural networks, and stacked autoencoders. All of these architectures are core components to deep learning with fundamentally different mechanics. They may be mixed-and-matched into interesting combinations depending on the type of the task at hand.

2.1.1 Restricted Boltzmann machines

Restricted Boltzmann machine (RBM) is a generative stochastic artificial neural network that learns features in an unsupervised manner based on a probabilistic model [51]. It is first invented under the name Harmonium by Paul Smolensky in 1986 [131]. Later, it was popularized by Geoffrey Hinton and his collaborators during the rise of machine learning when the group developed fast learning algorithms to effectively train RBMs. RBMs have been repeatedly used in a diverse area of applications as a means to reduce data dimension [144], solving classification problems [71, 72], feature learning [19], collaborative filtering [114], and topic modeling [152, 52, 135]. The success of RBMs largely owes to the fact that the extracted features being nonlinear in nature. Therefore, it often generates good results when used in conjunction with a linear classifier such as support vector machines (SVM) or a perceptron. Internally, an RBM attempts to maximize the likelihood of the data using a particular graphical
model, typically employing the learning algorithm via stochastic maximum likelihood. Using this method, it is capable of capturing persistent regularities from the data and learning a probability distribution over the set of the provided inputs. However, to effectively train the RBM model we require a sufficiently large dataset.

![Boltzmann Machine vs Restricted Boltzmann Machine](image)

**Figure 2.1:** Distinction between Boltzmann machines and restricted Boltzmann machines. In RBMs, there are no interconnectivity between units within the same layer (i.e. visible-visible or hidden-hidden connections).

Restricted Boltzmann machines are actually a variant of Boltzmann machines [1]. In a Boltzmann machine, nodes, or neurons in the same group are connected in addition to nodes from the other group (Fig. 2.1). Typically, two groups are present in a Boltzmann machines—they are often referred to as the visible or hidden units respectively. However, the interconnectedness of all the neurons within and between groups may complicate modeling. Hence, a restricted version of the Boltzmann machine is used with a constraint that the neurons must form a bipartite graph where there are no connections within a group. This restriction make learning easier, as the hidden units become conditionally independent given the visible states [13]. RBMs are also the building blocks to deep neural networks (DNN), where they can be stacked to increase the modeling capacity. The architecture will be discussed in Section 2.1.2.

**Architecture description for an RBM:** A standard form of RBM is binary valued where the hidden and visible units take values of either 0 or 1. Between the visible and hidden layers there is a weight matrix $W = (w_{i,j})$ of size $m \times n$ denoting the connection between visible unit $v_i$ and hidden unit $h_j$. In addition, there are bias terms $a_i$ and $b_j$ associated with the visible
units and hidden units respectively. With these notations in mind, the energy function \( E(v, h) \) of an RBM can be expressed as:

\[
E(v, h) = -\sum_i a_i v_i - \sum_j b_j h_j - \sum_i \sum_j v_i w_{i,j} h_j
\]

or more compactly, in vector form:

\[
E(v, h) = -a^T v - b^T h - v^T W h
\]

The joint probability distribution over hidden and visible units can now be defined in terms of the energy function as:

\[
P(v, h) = \frac{e^{-E(v, h)}}{Z}
\]

where \( Z \) is a normalizing factor called the partition function with the following form, defined as the sum of \( e^{-E(v, h)} \) over all possible configurations:

\[
Z = \sum_{v,h} e^{-E(v, h)}
\]

In other words, the normalizing factor \( Z \) ensures that the probability distribution sums to 1. The marginal probability of the visible vector is the sum over all possible hidden layer configurations with:

\[
P(v) = \frac{1}{Z} \sum_h e^{-E(v, h)}
\]

With no connections between the hidden and visible layers, activations of the hidden units are mutually independent given the activations of the visible units. Likewise, the visible unit activations are also mutually independent given the activations of the hidden units. Therefore, if we have \( m \) visible units and \( n \) hidden units, then we can write the conditional probabilities as:

\[
P(v|h) = \prod_{i=1}^m P(v_i|h), \quad P(h|v) = \prod_{j=1}^n P(h_j|v)
\]

Thus, the activation probabilities for each neuron is given by:

\[
P(h_j = 1|v) = \sigma \left( b_j + \sum_{i=1}^m w_{i,j} v_i \right), \quad P(v_i = 1|h) = \sigma \left( a_i + \sum_{j=1}^n w_{i,j} h_j \right)
\]

where \( \sigma \) is a sigmoidal function, such as a logistic sigmoid \( \sigma_1 \):

\[
\sigma_1(x) = \frac{1}{1 + e^{-x}}
\]
or the hyperbolic tangent function $\sigma_2$ which rescales the logistic sigmoid function to have an output range between -1 and 1:

$$
\sigma_2(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}
$$

**Training an RBM:** Restricted Boltzmann machines can be trained with the objective to maximize the product of probabilities assigned to a training set denoted by matrix $V$. The objective is to obtain a weight matrix $W$ such that:

$$
W^* = \operatorname{argmax}_W \prod_{v \in V} P(v)
$$

It will be simpler to express the equation as a sum rather than product. Equivalently, the algorithm aims to maximize the expected log probability of $V$ with:

$$
W^* = \operatorname{argmax}_W \mathbb{E} \left[ \sum_{v \in V} \log P(v) \right]
$$

Maximizing the expected log probability is equivalent to minimizing it. We thus define the loss function as the negative of the log probability, and aim to minimize it via the gradient descent algorithm using the update equations:

$$
\begin{align*}
W &:= W - \alpha \frac{\partial \log p(v)}{\partial W} \\
a &:= a - \alpha \frac{\partial \log p(v)}{\partial a} \\
b &:= b - \alpha \frac{\partial \log p(v)}{\partial b}
\end{align*}
$$

where $\alpha$ denotes the learning rate of the algorithm. The derivatives are expressed as follows [13]:

$$
\begin{align*}
\frac{\partial \log p(v)}{\partial W_{ij}} &= -E_v[p(h_i | v) \cdot v_j] + v_j^{(i)} \sigma(W_i \cdot v^{(i)}) + a_i \\
\frac{\partial \log p(v)}{\partial a_i} &= -E_v[p(h_i | v)] + \sigma(W_i \cdot v^{(i)}) \\
\frac{\partial \log p(v)}{\partial b_j} &= -E_v[p(v_j | h) \cdot v_j] + v_j^{(i)}
\end{align*}
$$
2.1.2 Deep neural networks

Deep neural networks (DNN) are obtained by stacking multiple layers or RBMs together. The more hidden layers are added to the network, the higher its nonlinear modeling capacity. Deep neural networks are also known as artificial neural networks (ANN), which are inspired by the observations and biological models proposed by Harvard neurophysiologists David H. Hubel and Torsten Wiesel. Hubel and Wiesel inserted a microelectrode into the primary visual cortex of an anesthetized cat and projected light and dark patterns on a screen in front of the cat. They found that some neurons of the cat fired rapidly depending on the angle of the presented lines. These neurons are referred to as simple cells. Further, they found that some other neurons (called complex cells) respond to the movement directions of the lines, which represents more complex stimuli. From these observations, they showed how the visual system builds complex representations from simple stimuli which established the fundamental concepts of deep learning [53, 148, 149, 112].

In deep neural networks, each layer of the neurons trains on different sets of features using the outputs from the previous layer. The deeper we advance into the network, the higher the complexity of the features that the network can recognize. For example, in the application of face recognition, the first layer of the network usually learns primitives such as simple edges and curves. As we move on to the intermediate layers, the hidden layers begin learning a combination of these primitives, such as the eyes, the mouth, the ears, and the nose. The deepest layers are capable of combining the parts and begin recognizing faces (see Fig. 2.2). This concept widely known as the hierarchy of features, that is, a hierarchy of features with increasing complexity and abstraction. Therefore, deep networks are suited for handling a large amount of very high-dimensional data sets.

**Architecture description for an DNNs:** In deep neural networks, the outputs of one layer become the input to the another layer. For the first layer, we have:

\[ x_2 = f(W_1v + b_1) \]

where \( x_2 \) denotes the output of the first layer and the input to the second layer, \( W \) is the weight matrix, \( v \) are the input data, and \( b \) is the bias. \( f(\cdot) \) is known as the activation function, and
Figure 2.2: Simple and complex features for faces, cars, elephants, chairs, and other objects. Figure courtesy of Lee et al. [74].

Figure 2.3: Stacking restricted Boltzmann machines, RBMs (a) to form deep belief networks, DBN (b). Adding a target layer on top will yield a deep neural network, DNN (c).

Modern deep nets prefer using the Rectified Linear Unit (ReLU) as the activation function over sigmoidal functions (e.g. the logistic sigmoid and tanh) due to the function being nonsaturating and provides good gradients conducive to training [97]. For all subsequent layers, we can generalize the equation to:

\[ x_{k+1} = f(W_k x_k + b_k), \quad k \geq 2 \]

**Supervised training of DNNs:** Weight updates can be done via the stochastic gradient descent algorithm using the following equation, similar to one used to train RBMs:

\[ \theta := \theta - \alpha \frac{\partial C}{\partial \theta} \]
In this equation, \( \theta \) denotes the model parameters which include the weights and biases of the layer. \( C \) is the cost function, or the loss function depending on the type of the problem we are attempting to solve. If the problem we are interested in is a classification problem, the zero-one loss and the negative log-likelihood loss are suitable candidates for the loss function. When using zero-one loss, a classifier is trained to minimize the number of errors on unseen examples. The loss can be written as:

\[
\ell_{0,1} = \sum_{i=0}^{\lvert \mathcal{D} \rvert} I_{g(x^{(i)}) \neq y^{(i)}}
\]

where \( \mathcal{D} \) is the training set, \( g \) is the prediction function with \( g : \mathcal{R}^\mathcal{D} \rightarrow \{0, ..., L\} \), \( I \) is the indicator function, and \( L \) is the number of output nodes corresponding to the class labels. The indicator function is defined as:

\[
I_x = \begin{cases} 
1 & \text{if } x \text{ is True} \\
0 & \text{otherwise}
\end{cases}
\]

and that:

\[
g(x) = \arg\max_k P(Y = k | x, \theta)
\]

What this means is that the predicted class is the class with the highest probability given the input. However, the zero-one loss is not a differentiable loss function. This may pose some issues when optimizing the model due to the problem with a non-smooth gradient where the computations may become overly expensive. Hence, we focus our attention on maximizing the log-likelihood of the classifier given the labels in a training set:

\[
\mathcal{L}(\theta, \mathcal{D}) = \sum_{i=0}^{\lvert \mathcal{D} \rvert} \log P(Y = y^{(i)} | x^{(i)}, \theta)
\]

Maximizing the log-likelihood is equal to minimizing the negative log-likelihood. Therefore, the loss function is rewritten as:

\[
\text{NLL}(\theta, \mathcal{D}) = -\sum_{i=0}^{\lvert \mathcal{D} \rvert} \log P(Y = y^{(i)} | x^{(i)}, \theta)
\]

If one wishes to perform regression using a deep neural network, then a natural choice of the loss function is to minimize the \( \ell_2 \)-norm (for example) of the difference between the output of the neural network \( \hat{y} \) with the target vector \( y \):

\[
\theta^* = \arg\min_\theta \| y - \hat{y} \|_2
\]
Unsupervised greedy layer-wise pretraining of deep belief networks (DBN): When training a DNN, the model parameters are initialized randomly as means of symmetry breaking. However, randomly initialized weights may slow training and cause the model to converge to bad local optima. To combat this, the deep neural network can be pretrained in a greedy layer-wise manner without labeled data (hence it is a deep belief network rather than deep neural network), using the same algorithm used to train an RBM [36]. After the first layer is trained, we keep the weights and biases of the first layer constant. The transformed input from the layer is utilized to train the next layer. This process is repeated for the desired number of layers in the network with each iteration propagating either the samples or mean activations to higher levels. As training continues, the product of probabilities assigned to the input is maximized. Once all the layers are trained, the pre-trained model parameters are finetuned via supervised backpropagation.

2.1.3 Convolutional neural networks

Deep neural networks have demonstrated success in image recognition. However, they quickly suffer from the curse of dimensionality if the input data dimensions is large. For example, recognizing a $32 \times 32$-pixel color (RGB) image from the CIFAR-10 dataset will result in the first layer having width x height x channels = $32 \times 32 \times 3 = 3,072$ weights in the first layer. Practically, most inputs in image recognition applications are much larger than $32 \times 32$ pixels. For example, a $512 \times 512$-pixel RGB image will now result in $512 \times 512 \times 3 = 786,432$ weights. Hence, we observe that the framework becomes intractable with larger input dimensions.

There are a few drawbacks to using DNNs in such cases. Firstly, the large number of fully connected layers will result in an excessively huge model when used in conjunction with a multi-layered deep network. Inferencing will take a longer time due to the increased computational complexity. Secondly, fully connected layers do not take into account the spatial structure of the image data and do not consider the local correlation of the features in the context of a two-dimensional image. Consequently, it places equal weighting on pixels that are close and far apart. Full connectivity clearly wastes valuable computational resources. Furthermore, the large number of parameters will be difficult to train and makes the network prone to overfitting.
Convolutional neural networks (CNN) [65] are fundamentally different than regular DNNs in terms of architecture. It mitigates the issues posed by DNNs by exploiting the local 2D-correlations in input images, thus making it a very attractive model for working with image data. Unlike DNNs, neurons within a layer of CNN (called a receptive field) are only connected to a small region of the previous layer, as shown in Fig. 2.5. This ensures that the learned filters generates the strongest response according to a spatially local input pattern. By stacking multiple layers of filters together, the modeling capacity of the architecture becomes increasingly non-linear and captures more global features. Hence, at deeper levels the model can assemble lower-level features to form higher-level features.

**The convolutional layer:** Convolution is a sliding function (a filter) applied to the 2D input matrix. The convolutional layer is the most important building block of a CNN and consists of filters (or kernels) that can be learned. These filters have small receptive fields to extract local structures and uses shared weights (Fig. 2.5). With a set of different filters, the filters may specialize in detecting different features—the activations of different filters will depend on the specific type of feature at some spatial location in the input image. As the filters are convolved over the input, feature maps are generated (as illustrated in Fig. 2.6). Stacking the feature maps along the depth dimension gives rise to the full output volume of the convolutional layer.
Figure 2.5: The use of shared weights reduces memory footprints, preserves local correlations, and reduces the number of learnable parameters to prevent overfitting. The colors of the sparse connections denote shared weights.

Figure 2.6: Convolving a filter (orange) over the input image (blue) to generate a convolved feature map (green). For each output, the values of the filters are multiplied element-wise with the overlapping region in the input image. The products are summed up as one output element in the convolved feature.

There are three most prominent hyperparameters that control the size of the output volume. With a deep layer, many neurons connect to a particular region in the input volume. The neurons will learn to activate different depending on the features in the input. For instance, neurons in the same layer may become active in the presence of edges in different orientations and lie in the same feature hierarchy. Large convolution strides will result in a smaller feature map due to less overlapping receptive fields, whereas small convolution strides (such as $1 \times 1$) convolution will result in strongly overlapping receptive fields and, subsequently, larger feature maps. In all cases, the dimensions will be reduced depending on the size of the convolution strides. However, it is sometimes desirable to preserve the original spatial dimensions of the input volume. To achieve this, the borders of the input volume can be padded with zeros (termed as zero-padding) such that after the convolution operation, the reduced spatial dimensions are compensated to maintain original dimensions.

The pooling layer: Pooling is a form of nonlinear downsampling. Common practices employ the maxpooling scheme, where a $2 \times 2$ matrix for example is downsampled into a $1 \times 1$ value by selecting the element with the highest value in the matrix. Of course, other pooling
functions exist too; one may downsample by averaging the values or even computing the $\ell_2$-norm. Pooling is useful because it removes redundancies and helps reducing the dimensions of the data. The intuition is that once a feature is detected, the exact location of the feature may not be as important as the approximate location relative to other features. Doing so also reduces the number of learnable parameters to combat overfitting as well as reducing computation time. Note that using pooling layers is up to the discretion of the user; it is a common practice to periodically insert a pooling layer after several convolutional layers. An additional benefit that pooling offers is the translation invariance of features. However, most studies are gravitating towards using smaller filters [44] and discarding the pooling layer [133] in order to prevent an excessively aggressive reduction in dimension.

**The fully connected layer:** After several reductions in dimensions from convolution and pooling, the location information of the features become less important. Hence, we can connect the feature maps generated by the filters to the fully connected layers to increase modeling capacity. We can think of the feature maps being vectorized as an input to a single neural net layer. From this point onwards, the forward pass will be similar to the procedure outlined in Section 2.1.2. In the context of classification, a softmax function can be applied on the sigmoid activations of each output neuron to obtain a probability distribution, where the class with the highest probability is selected as the class prediction. Similarly, one can choose to minimize the loss function (such as negative log-likelihood) and optimize the model parameters via gradient descent.
2.1.4 Stacked autoencoders

An autoencoder tries to learn the approximation to the identity function in an unsupervised manner, such that the reconstruction of the input is similar to the actual input. In unsupervised learning, only unlabeled data is used. At first glance, it seems to be trivial to learn the identity function. However, this problem becomes not so trivial anymore if we impose some constraints to the learning process such that the algorithm can discover interesting or meaningful underlying patterns in the input data (Fig. 2.8).

![Diagram of autoencoder](image)

Figure 2.8: Learning the identity function with an autoencoder (i-input, h-hidden unit, o-output). (a) The hidden layer is an identity function where the input can be fully reconstructed without loss. (b) Reduced number of hidden units captures the most important features to effectively reconstruct the input. (c) Sparsity constraints suppress activations of the hidden units to help extracting low-dimensional features from the input.

As a concrete example, suppose we are given an image with $10 \times 10$ pixels for its dimensions. Flattening this 2D image results in a 100-dimensional vector which corresponds to 100 input units into the autoencoder. If the autoencoder only has 50 units for the hidden layer, we are now forced to learn a compressed representation of the input such that the 100-dimensional output can be reconstructed from only the activations from these 50 hidden units. Note that if the input data is i.i.d. random, then discovering the underlying structure would be tremendously difficult because the data is essentially a white-noise signal—there are no prominent structures associated with the data. In reality, the data that we frequently encounter have structures and the inputs are often correlated. To illustrate, two adjacent pixels in a natural 2D image are
often correlated in terms of color. Objects in the image will also have a series of ‘pixels’ that form the edges of the object. Autoencoders are algorithms that can automatically discover these correlations. Similar to principal component analysis (PCA), autoencoders can learn low-dimensional representation of the inputs by capturing the codes within the data; in fact, the optimal solution to an autoencoder is strongly related to one found with PCA if linear activations or only a single sigmoid hidden layer are used [21]. The added advantage of autoencoders is that they can be stacked to form stacked autoencoders (SAE), another deep architecture that has a superior nonlinear modeling capacity compared to a single layer of autoencoder or PCA [146].

![Diagram of denoising autoencoder](image)

Figure 2.9: Training a single layer of denoising autoencoder. Input vector $x$ is corrupted to obtain $\tilde{x}$. The objective is to learn the parameter $\theta$ that is able to reconstruct $z$ from a compressed representation of $\tilde{x}$ such that the error between $z$ and $x$ is minimized. The same parameter $\theta$ is used in both the encoding and the decoding phase.

Formally, an encoder is a deterministic mapping denoted by $f_\theta$ (parameterized by model parameter $\theta = \{W, b\}$ where $W$ and $b$ are the weights and biases) transforms the input vector $x$ into a representation $y$. The decoder inversely maps the hidden representation $y$ into the reconstructed input $z$ with the function $g_\theta^T$. The idea is to learn the model parameters $\theta$ for both mappings such that the error $L_H(x, z)$ between the reconstructed input and the original input is minimized. Suitable loss functions could be cross-entropy loss with an affine-sigmoid decoder, or the mean squared error (MSE) loss with an affine decoder.
A denoising autoencoder performs an additional step on the input vector \( x \) by applying a salt-and-pepper noise to form \( \hat{x} \). In other words, some elements in the vector \( x \) are randomly set to be zero. Therefore, in addition to learning a compressed representation of the input, a denoising autoencoder tries to further recover the clean input from the noisy input. It has been shown in [145] that applying a random corruption to the input will enable more robust extraction of features.

![Figure 2.10: Stacking multiple autoencoders to form a deep architecture. The layers are trained separately.](image)

Autoencoders can be stacked on top of each other to form a deep architecture. After learning the encoding function of the first layer, the subsequent layers can be trained in a layer-wise manner while keeping the parameters of the previous layers fixed. This process is repeated for all layers in an unsupervised manner to extract robust features from the training set. When the procedure has been completed, supervised learning can take place by training the entire network by treating it as a regular DNN or adding an SVM classifier/logistic regression on top.

### 2.2 Regularization

In this subsection, we discuss some regularization techniques used in our research to combat the problem of overfitting. When a model overfits, the generalization power of the model to make inference based on unseen data is greatly reduced. An overfitted neural network is capable of reproducing even the finest details of noise in the input dataset. In other words, the bad model reproduces a non-smooth function corresponding to the non-smooth nature of the noise. As another method to combat overfitting, here we will also introduce the early stopping algorithm to the readers.
2.2.1 Weight decay

Weight decay is a form of regularization (Fig. 2.11). In essence, the weights within a neural network are constrained such that no one weight parameter dominates the other to ensure a smooth output function. When performing weight decay, we can modify the loss function that is minimized during the training procedure. Let \( L \) denote the original loss function. The modified loss function, \( \tilde{L} \) is therefore:

\[
\tilde{L} = L + \lambda S
\]

where \( \lambda \) is the regularization parameter (a constant that controls the level of smoothing) and \( S \) is the value obtained after a chosen function is applied on the weights. For example, in \( \ell_2 \) regularization, the weight vector is squared. With the modified loss function, the optimizer penalizes the loss function if the model weights become too large. On the other hand, \( \ell_1 \) regularization may achieve the same effect while possibly attaining sparsity—a situation where some of the weights become zero. Neurons with \( \ell_1 \) regularization end up using a subset of the most important inputs and refrains from using noisy inputs.

2.2.2 Early stopping

Early stopping is another form of regularization used to prevent overfitting when training a model using an iterative method such as stochastic gradient descent. This method is used in conjunction with cross-validation, where the training dataset is further split into a training set and a validation set. The model learns the parameters using the training set, and reevaluates
the model performance on the validation set after each update iteration. Generally, the training error will keep decreasing as the model is being trained. However, the plot of validation error will generally exhibit a U shape, meaning that at one point the validation error will keep increasing even though the training error continues to decrease (see Fig. 2.12). When the errors diverge, the model is said to be overfitting. In early stopping, we select the saved model where validation error is at minimum.

![Early stopping algorithm](image)

**Figure 2.12:** The early stopping algorithm prevents overfitting by evaluating losses on both the training set and the validation set.

In some cases, the validation error may increase temporarily before decreasing again and signifies that the model has escaped bad local optima. Some algorithms include other user-defined parameters such as *patience* and *improvement threshold* that control how many iterations that the model should be trained. A large value for patience will allow the model to keep training for some extended time, even if the validation error continues to increase, with the hope that it will escape bad optima. On the contrary, it may be time-consuming to wait for the convergence of the validation error. The early stopping algorithm may stop training when the decrease in validation error is lower than the user-defined improvement threshold.

### 2.3 Feature visualization

One of the main claims of a hierarchical semantic feature extraction tool is that meaningful patterns in the data can signify the underlying characteristics of the process. Therefore,
visualizing the learned features is crucial to both understand and verify the performance of the feature extractor. Furthermore, intermediate feature visualization may lead domain experts to scientific discoveries that are not easy to figure out via manual exploration of large volume of data.

For the lowest layer in fully connected deep networks (e.g. DBN and DNN), plotting the weight matrix may be sufficient to visualize the features learned by the first hidden layer. Since the dimensionality of the input and the weights are in the same order, the vectors of weights for each input can be reshaped into the dimension equal to the resolution of the input image. Thus, the visualizations are usually intelligible. Complexity arises for visualizing features learnt at deeper layers because they lie in a different space from the visible data space. At the same time, the dimension of the weight matrix depends on the number of hidden units between the layer and the layer before. Thus, plotting the weight matrix will result in an incomprehensible visualization which typically resembles the appearance of white noise. To obtain filter-like representations of hidden units in the DBN, a recent technique known as Activation Maximization (AM) is used [38]. This technique seeks to find inputs that maximize the activation of a specific hidden unit in a particular layer and the technique is treated as an optimization problem. Let $\theta$ denote the parameters of the network (weights and biases) and $h_{ij}(\theta, x)$ be the value of the activation function $h_{ij}(\cdot)$ (usually the logistic sigmoid function) of hidden unit $i$ in layer $j$ on input $x$. Assuming the network has been trained, $\theta$ remains constant. Therefore, the optimization process aims to find

$$x^* = \arg\max_{x\text{ s.t. } ||x||=\rho} h_{ij}(\theta, x)$$

where $x^*$ denotes the inputs that maximizes the hidden unit activation. Although the problem is a non-convex optimization problem, it is still useful to find the local optimum by performing a simple gradient ascent along the gradient of $h_{ij}(\theta, x)$ because in many cases, the solutions after convergence are able to visualize the patterns of the inputs that are being learned by the hidden units.

In CNNs, learned features can be visualized by plotting the weight matrix of the feature maps in the same manner as obtaining filter-like representations from DNNs.
Summary. We have presented various deep learning architectures and regularization techniques that are used in our studies. The following chapters are dedicated to our published/submitted papers describing how we solve engineering problems with deep learning in detail along with additional results.
PART I

DEEP LEARNING FOR DECISION MAKING IN MECHANICAL SYSTEMS
CHAPTER 3. DATA-DRIVEN DEEP LEARNING MODELS FOR EFFICIENT DESIGN OF MICROFLUIDIC FLOW PATTERNS

3.1 Motivation

In this chapter, the application of deep learning in design engineering (specifically, microfluidic device or lab-on-a-chip design) is explored, which has a large implication on manufacturing processes, chemical engineering, and biology. The main goal of this study is to investigate the potential of hierarchical feature extraction techniques to become scalable design tools via learning semantic representations from a relatively small number of training examples. Apart from this objective, we fused multiple deep learning architectures to achieve improved performance and compared supervised and unsupervised schemes to explore the impact of training data generation technique on the feature extraction process.

Controlling the shape and location of a fluid stream provides a fundamental tool for creating structured materials, preparing biological samples, and engineering transport of heat and mass. Previous methods to manipulate the cross-sectional shape and flow deformation of fluids have focused on creating chaos and mixing by fluid twisting instead of ordering and structuring streams with precise sequences of fluid perturbations. The concept of sculpting fluid streams in a microchannel using a set of pillars that individually deform a flow has been recently discovered and demonstrated with great success [7]. Pre-computed deformations from individual pillars can be used in sequence as operations on fluid flow, enabling rapid simulation of arbitrary microfluidic devices. In this way, an elegant mathematical operation can yield the final flow shape for a sequence without an experiment or additional numerical simulation.

Although this approach has allowed for the sculpting of complex fluid shapes [137], creating user-defined flow shapes for practical applications currently requires laborious and time-consuming
trial and error design iterations. The ability to create a user-defined flow shape and automatically
determine a sequence of pillars that yields this shape is a significant and impactful advance.
While standard techniques that reformulate this inverse design problem into an unconstrained
optimization problem [136] have been successful, they are invariably time-consuming. This
makes their utility in real-time design suspect.

With this motivation, the applicability of deep learning models is explored to serve as a
map between user-defined flow shapes and the corresponding sequences of pillars. Previous
works have used evolutionary algorithms and neural nets to solve forward computational fluid
problems [96, 33, 11, 8], but to the best of the authors’ knowledge, this is the first application of
deep learning based hierarchical feature extraction to solve the inverse problem specifically in
fluid mechanics.

The major contributions of the study are outlined below:

1. A novel ‘hierarchical feature extraction for design’ approach using deep neural networks
   (DNN) [49, 48] and convolutional neural networks (CNN) [65] is proposed to capture multi-
scale patterns of a deformed fluid flow to solve a simultaneous multi-class classification
problem. We show that such models can learn good representative features even from
a very small subset of the entire design space (can be as skewed as 150,000 out of $10^{15}$
possibilities) leading to a scalable design tool.

2. Using domain knowledge regarding the massive nonlinear design space, a quasi-random
   sampling method is proposed based on a metric on the output space to generate robust
   training data for improved feature extraction.

3. The feasibility of the proposed approach is tested and validated via experiments with
different network structures (supervised and unsupervised), with results showing com-
petitive prediction accuracy and significant reduction in execution time compared to the
state-of-the-art genetic algorithm based design tool.

4. Hierarchical features are visualized to extract coherent structures in flow which may lead
to better understanding of the physics of complex flow patterns.
5. The proposed method provides near real time design that greatly enables design exploration by the end user. As conventional state-of-the-art methods (such as genetic algorithm) take dozens of hours to identify one appropriate device design, multiple design iterations result in a design time of weeks for a microfluidic device. This wait time makes CAD like design/analysis of microfluidic devices infeasible and bottlenecks creative exploration especially for biomedical applications. We show that hierarchical feature extraction is able to address this issue.

6. A formulation of learning causal shape transformations to predict a sequence of transformation actions is presented, in the setting where only an initial shape and a desired target shape are provided.

7. An integrated hierarchical feature extraction approach using stacked autoencoders (SAE) [145] with convolutional neural networks (CNN) [65] is proposed to capture transformation features to generate the associated sequence resulting in the transformation.

8. The proposed approach is tested and validated via numerical simulations on an engineering design problem (i.e. flow sculpting in microfluidic devices), with results showing competitive prediction accuracy over previously explored methods.

Our collaborators intend to utilize this framework for concrete biomedical applications that require microfluidic device design, for example: (a) Device design to move fluid surrounding cells (lymphoid leukemia cells) against a far wall of the microfluidic channel where it can be collected at high purity while cells are maintained in the channel center. High purity collection can allow reuse of valuable reagents that are used to stain cells for diagnostics. (b) Device design to wrap fluid around the microchannel surface. This will be used to characterize binding of p24 (an HIV viral capsid protein) to anti-p24 antibody immobilized on the microchannel surface. The idea of flow sculpting here is to enhance reaction of low abundance proteins that can improve diagnostic limits of detection for various diseases. Therefore, this novel study may open up many new application areas for the machine learning community related to thermo-fluid sciences and design engineering.
3.2 Problem setup

The ability to control shape and location of fluid streams allows the capability of creating structured materials, preparing biological samples, and engineering heat and mass transport by ordering and structuring streams with precise sequences of fluid perturbations. Using microchannels populated with a set of pillars, the pillars can individually deform a flow to achieve the final desired shape [7] (Fig. 3.1). Pre-computed deformations from individual pillars have been experimentally validated and enabled rapid simulation of an arbitrary microfluidic device in the order of seconds. Therefore, this mathematical operation (referred to as the forward problem) can elegantly yield the final flow shape for an arbitrary sequence without additional numerical simulation.

![Figure 3.1: Pillar programming. Each pillar contributes to the deformation of the flow. The position-diameter pair of each pillar is assigned an index which will be used as class labels for classification.](image)

However, practical applications require solving the inverse problem, that is, to generate a sequence of pillars given a user-defined flow shape. Without intelligent computer algorithms, such tasks require time-consuming trial and error design iterations. The automated determination of pillar sequences that yields a custom shape is a significant and impactful advance. Although works have been done to frame this inverse problem as an unconstrained optimization problem [136], they are invariably time-consuming. While many methods are used to solve the forward problem [96, 33, 11, 8], only a limited amount of effort has been done in solving the inverse problem [84]. For practical and time-efficient applications, deep learning methods are explored to map user-defined flow shapes and the corresponding pillars of sequence.
In this work, we use DNNs and CNNs to extract pertinent nonlinear features from flow shape images as well as learn a classifier with pillar configurations as the target classes. Deep learning tools are ideal in this problem as configuring hand-crafted features for the flow shape images is an extremely nonintuitive and tedious process. As the forward problem of generating simulated flow shapes from a sequence is computationally inexpensive and can be done in a fraction of a second, a large set of labeled images (in the order of thousands/millions) can be generated for training. The class labels (in the order of hundreds) represent different pillar sequences determined by the number, size, and location of pillars. Training is performed using Theano platform [16] on NVIDIA TITAN Black GPU.

3.3 Related approach: Genetic Algorithm

Previous work has attempted to identify pillar sequence(s) for a given fluid flow shape using the current state-of-the-art genetic algorithm (GA) [136]. The GA is an evolutionary search heuristic that solves optimization problems using techniques inspired from natural evolution. Broadly, a random initial population of pillar sequences is generated. This population is evolved through crossover and mutation toward a targeted flow shape by penalizing sequences that produce flow shapes different from the target, and rewarding those that are similar. These punishments and rewards are borne out in the breeding process between generations. After many generations of breeding, the population should produce a larger subset of flow shapes similar to the target shape than the initial population.

3.4 Sequence prediction by Simultaneous Multiple Classification

One approach to solve the inverse design problem is to assign class indices to pillars with different specifications. For example, a pillar at position 0.0 and diameter 0.375 will be assigned an index of 1, whereas another pillar at position 0.125 and diameter 0.375 will be assigned an index of 2. Index assignment is performed over a finite combination of pillar positions and diameters that has been obtained by discretizing the design space. In this study, there are 32 possible indices that describes the diameter and position of a single pillar. Consequently, this
becomes a 32-class classification problem for one pillar. In general, when presented with a flow shape that is generated from an $n_p$-pillar sequence, the trained model needs to simultaneously predict the class for each of the $n_p$ pillars to form a sequence of indices from which the positions and diameters for each pillar are retrieved. Essentially, instead of solving a single classification problem just like what that has been typically done, the model actually solves one problem for each pillar based on the same weights and biases that have been learned by the model. This framework is noted as Simultaneous Multi-class Classification (SMC).

Note, it may seem that assigning classes to target pillars suffers from under-specification. Actually, this was just a user-defined design constraint as there are limited configurations for individual pillar placements based on manufacturability.

This formulation is applicable on both DNN and CNN architectures (CNN with SMC shown in Figure 3.2) and requires only a slight modification in the loss function. For an $n_p$-pillar problem, the loss function to be minimized for a data set $\mathcal{D}$ is the negative log-likelihood defined as:

$$
\ell_{\text{total}}(\theta = \{W, b\}, \mathcal{D}) = - \sum_{j=1}^{n_p} \sum_{i=0}^{|\mathcal{D}|} \log \left( P(Y = y^{(i)} | x^{(i)}, W, b) \right)
$$

where $\mathcal{D}$ denotes the training set, $\theta$ is the model parameters with $W$ as the weights and $b$ for the biases. $y$ is predicted pillar index whereas $x$ is the provided flow shape test image. The total loss is obtained by summing the losses computed for each pillar.

Figure 3.2: CNN with the SMC problem formulation.
3.4.1 Training data generation using a metric on the output space

As the number of pillars increases, number of design possibilities increases exponentially. Although training data generation process is fairly quick, the design process has to be efficient with a relatively much smaller number of training examples in order to be scalable. To investigate the scalability of the proposed technique, the number of training data examples is kept constant even as the number of pillars increases. In this study, each training dataset consists of 150,000 flow shape images generated from pillar sequences spanning 4-10 pillars. However, this means that each set of training data covers a vanishingly small fraction of the possible design space, with coverage shrinking exponentially as the number of pillars in a sequence increases (see Table 3.1 and Figure 3.8 (a)). Ideally, each dataset would uniformly sample the design space and capture a wide variety of features, but high dimensionality with longer pillar sequences makes this difficult for a uniform distribution. Randomly sampling would likely have clustering or gaps in coverage as the number of pillars in a sequence increases. We therefore use quasi-random Sobol sequences to more uniformly cover the high dimensional space [132]. Unlike typical pseudo-random number generators, a Sobol sequence enforces low discrepancy sampling. Discrepancy, $D_N$, is a measure of coverage for a sampled set $P$ of size $N$ within a set of intervals $B$ spanning the available space, as defined by [67]:

$$D_N(P) = \sup_B \left| \frac{A(B; P)}{N} - \lambda(B) \right|$$

(3.1)

where $A(B; P)$ is the number of sampled points that fall into the intervals defined by $B$, and $\lambda(B)$ is the Lebesgue measure of $B$ (i.e., the size of the design space). By minimizing $D_N$ (low discrepancy), the points sampled will fill the space defined by the intervals in $B$ more evenly than uncorrelated random points [104]. Thus, low discrepancy sampling helps to prevent crowding or large gaps in the selected pillar sequences, which should create a more balanced and varied dataset of pillar sequences as illustrated in Figure 3.3.

Sobol sequences create intervals by successively halving the size of the domain as the number of sampled points increase. Each new point in the sequence is therefore based on the previous point, which leads to the quasi-random nature of the Sobol set, as the sampling method is ultimately deterministic. The sampling used here chooses different pillar configurations from
Figure 3.3: Uniform sampling vs. Quasi-random sampling. A set of 160 2-pillar sequences are formed using uniform (left) and Sobol (right) sampling methods. The first 10 sampled sequences are shown in red, the next 50 sampled sequences in green, and the final 100 sampled sequences in blue.

<table>
<thead>
<tr>
<th>$n_p$</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n_s$</td>
<td>$32^4$</td>
<td>$32^5$</td>
<td>$32^6$</td>
<td>$32^7$</td>
<td>$32^8$</td>
<td>$32^9$</td>
<td>$32^{10}$</td>
</tr>
<tr>
<td></td>
<td>$1.05(10^6)$</td>
<td>$3.36(10^7)$</td>
<td>$1.07(10^9)$</td>
<td>$3.43(10^{10})$</td>
<td>$1.10(10^{12})$</td>
<td>$3.52(10^{13})$</td>
<td>$1.12(10^{15})$</td>
</tr>
</tbody>
</table>

Table 3.1: Number of possible sequences, $n_s$ in an $n_p$-pillar sequence.

a set of 32 pillar types, with 4 diameters and 8 locations in the channel. Multiple pillars in a sequence means that the space being sampled is a hypercube of dimension $n_p$. Training, validation, and testing data are formed by disjoint Sobol sequences. The flow shape images are then formed using the same method as in [136], which reduces fluid flow simulation to simple matrix multiplication.

Remark 3.4.1 In this training data generation process, we use the domain knowledge of the output space i.e., the space of the pillar sequence arrangements. With a simple Euclidean metric on the output space, the proposed quasi-random sampling process helps extract representative flow shape features from the massive design space. Such a method can have a broader implication for learning hierarchical features from physical datasets (both simulated and experimental) in general.
3.4.2 Experiments and results

This section reports the parameters of the network and evaluates the performance of both DNN (with unsupervised pre-training using a DBN) and CNN in solving the problem.

3.4.2.1 Network parameters and description

The DNN contains 5 hidden layers with 2,000 hidden units each. The input is an image of the flow shape with dimensions of $24 \times 200$ pixels flattened into a $1 \times 4800$ row vector with binary pixel intensity values of 0 (black) and 1 (white). The CNN consists of 2 convolutional layers (20 s of size $11 \times 11$ for first layer, 50 filters of size $4 \times 4$ for the second layer), 2 pooling layers (downsampled by $2 \times 2$ maxpooling), and one fully-connected layer with 500 hidden units. Training was performed on 150,000 samples (with an additional 20,000 validation samples) and took approximately 6 hours while testing was performed on 20,000 samples for each set of different sequence length. The learning rate is 0.1. All hyperparameters mentioned above have been cross-validated and chosen based on repeated experiments.

3.4.2.2 Performance of CNN-SMC

![Figure 3.4: Eight example predictions ($n_p = 4$). The left side of each pair is the target flow shape and the right side is the flow generated from the predicted pillar sequences.](image)

After the network makes a prediction, the resulting sequence is used to generate the flow shape which is then compared with the target shape as shown in Figure 3.4. The pixel match rate (PMR) is used as a performance metric. In simpler terms, given an original flow shape image and an image generated from the predicted pillar sequences using the DL model, two
Table 3.2: PMR-based performance for DNN vs CNN. The appended number 4 is the number of pillars in the sequence.

<table>
<thead>
<tr>
<th>PMR (%)</th>
<th>DNN-4</th>
<th>CNN-4</th>
<th>Test Samples (%)</th>
<th>DNN-4</th>
<th>CNN-4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Min</td>
<td>56.90</td>
<td>52.83</td>
<td>≥ PMR 80</td>
<td>65.17</td>
<td>84.33</td>
</tr>
<tr>
<td>Mean</td>
<td>82.60</td>
<td>88.09</td>
<td>≥ PMR 85</td>
<td>49.59</td>
<td>70.89</td>
</tr>
<tr>
<td>Median</td>
<td>84.88</td>
<td>89.92</td>
<td>≥ PMR 90</td>
<td>27.20</td>
<td>49.50</td>
</tr>
<tr>
<td>Max</td>
<td>100.00</td>
<td>100.00</td>
<td>≥ PMR 95</td>
<td>5.81</td>
<td>18.47</td>
</tr>
</tbody>
</table>

Table 3.3: PMR-based performance for CNN with 7-pillar sequence that is trained with randomly-generated training data vs. quasi-randomly generated training data.

<table>
<thead>
<tr>
<th>PMR (%)</th>
<th>CNN-7, Random</th>
<th>CNN-7, Quasi-random</th>
</tr>
</thead>
<tbody>
<tr>
<td>Min</td>
<td>44.62</td>
<td>50.36</td>
</tr>
<tr>
<td>Mean</td>
<td>76.97</td>
<td>78.44</td>
</tr>
<tr>
<td>Median</td>
<td>77.46</td>
<td>79.44</td>
</tr>
<tr>
<td>Max</td>
<td>98.11</td>
<td>97.58</td>
</tr>
</tbody>
</table>

pixels at corresponding locations match if they both have the same color. Formally,

\[
PMR(\%) = 100\% \left( 1 - \frac{||p - \hat{p}||_1}{|p|} \right) \quad \text{or} \quad PMR = \left( 1 - \frac{||p - \hat{p}||_1}{|p|} \right)
\]

where \(p\) is the target image vector, \(\hat{p}\) is the predicted image vector, and \(|p|\) denotes the number of elements in the vector (i.e., the total number of pixels in the image).

Table 3.2 shows the PMR using both DNN and CNN architectures for a 4-pillar sequence. CNN clearly achieves a higher average similarity compared with DNNs and is able to predict the pillar sequence that generates a perfectly matching flow shape with the desired flow shape (i.e., PMR = 100%). With CNN, 84.33% of the test samples scored a PMR of 80% and above which is much higher than using DNN. As the PMR threshold is increased, the performance of CNN remains superior than the other by a significant margin.

Table 3.3 shows the comparison of PMR score when the CNN is trained with a quasi-randomly generated training data (using Sobol sequences) with the expected PMR score with randomly generated training set. The model trained with quasi-randomly generated training data is able to generalize better and predict more accurate sequences.
Remark 3.4.2 As feature extraction concerns extracting informative, non-redundant features that facilitate the generalization procedure, efforts are usually placed to improve training algorithms or reframing the problem such that a different feature extraction technique can be applied. Better performance from using quasi-random training data suggests that domain knowledge should not be overlooked and can be incorporated into the data generation procedure to aid effective learning even with datasets severely limited in size.

Remark 3.4.3 What is a good PMR? Finding out the percentage of samples higher than a specific PMR threshold is particularly useful for design purposes. In the context of manufacturing, 84.33% of the predictions from the CNN is satisfactory if the design error tolerance is 20%. For biomedical applications as described in the introduction, a PMR of at least 80-85% is good enough to make reasonably efficient devices. Minor variations in shapes are allowable, as fluid diffusion anyway results in shape distortion. Essentially, this tool serves as the first step in determining pillar sequences within acceptable criteria without investing too much time in design iterations.

3.4.2.3 Visualization of learned features

This section presents the visualization of learned features for both network structures using Activation Maximization outlined in Section 2.3. Visualizing features shows that the network can extract meaningful features in the flow shapes that can be interpreted by the domain experts. Figure 3.5 visualizes the input image vectors that maximize activations of the hidden units. While lower hidden layers present a collection of structures that contributes to the overall flow shape, the final hidden layer shows identifiable flow shapes that are captured by the hidden units. Figure 3.6 shows the filters and convolved feature maps from both convolutional layers. Filters, with corresponding feature maps, are chosen at random to reveal the edge-like features that are learned by the model.

Remark 3.4.4 The visualizations show that the hierarchical feature extraction processes can learn coherent structures in the flow shapes that can lead to better understanding of ingredients of the overall flow and hence the physics of complex flow patterns. For example, for different pillar sequences, changes in activation patterns (projected down to the visible layer) can signify
fundamental implications of pillars on the flow shapes. In this context, although fully supervised CNN has a higher performance accuracy, it is quite difficult to project activation patterns down to visible layer as it involves (memory and computationally) expensive de-convolution process [155]. On the other hand, DNN visualizations shown here demonstrate various features learned at different levels without large complexity. For instance, visualization of the 5th layer provides valuable insights to the diversity of flow shapes achievable through different pillar sequences.

3.4.2.4 Comparison of CNN-SMC with genetic algorithm

This section compares the PMR and execution time between the state-of-the-art GA and deep learning (DL) methods (using only CNN-SMC) for sequences with more than four pillars. Among the four examples shown in Figure 3.7, GA gives marginally better predictions in terms
of quantitative accuracy. However, the DL outcomes are generally acceptable as the errors fall within the typical design threshold for this application.

On the other hand, as an evolutionary heuristic, the GA requires many fluid flow shapes to be produced and evaluated during a search. Therefore, the execution time for identifying a pillar sequence for a single image can take hours [136]. However, a trained deep network will give results with a runtime on the order of seconds. Furthermore, the choice of cost function will greatly impact the effectiveness of a GA search, and therefore requires laborious tuning. Given the time required for a single run, this tuning process can be considerably more complex and difficult than the training of a deep network. While the algorithm runtime using DL methods mentioned in the previous section includes the time needed to load the model and predict six different pillar sequences ($n_p = 5, 6, ..., 10$) given an input, prediction actually occurs almost instantaneously since most of the time spent is used to load the model weights. Hence, DL methods allow for near real-time flow sculpting. More importantly in this case, the user could manipulate a flow shape using a graphical interface that would elicit key expert knowledge.

![Figure 3.7: Comparison of pixel match rate and algorithm runtime for GA versus DL methods (using CNN-SMC) to generate pillar sequences that has the best shape reconstruction closest to the target flow shape. Four examples are provided.](image)

Another interesting aspect is that the number of possible pillar configurations (hence the size of the search space) increases exponentially with the number of pillars. With the training set kept constant at 150,000 samples in this current study, the ratio of training samples to the number of all possible pillar sequences shrinks very quickly with increasing pillar sequence length.
Figure 3.8: (a) Variation of the average PMR and training size ratio (ratio of training samples to the number of all possible pillar sequences, on a logarithmic scale) with increasing number of pillars in a sequence. (b) Flow shapes generated from predicted pillar sequences.

Figure 3.8 (a) shows the capability of the hierarchical model under this “curse-of-dimensionality” challenge. Even with an exponential decrease in coverage of the design space, the PMR rate only degrades linearly. For example, 68% PMR is achieved (may be slightly less than acceptable) using only 150,000 training examples out of $10^{15}$ possible pillar configurations for the 10-pillar sequence. Figure 3.8 (b) shows the flow shape generated from predicted pillar sequences based on an example input flow shape. The PMR values for this example are consistent with the results obtained from Figure 3.8 (a). In the design context, the user will be presented with a selection of outputs to choose from. In this use case, the user may accept a small decrease in accuracy (95.88% to 94.52%) to gain possible financial savings by using fewer pillars (5 instead of 6).

3.5 Action sequence learning for causal shape transformation

The framework proposed in the previous section is not without drawbacks. The pillars are predicted in a joint manner, where each of the pillars in the sequence are inferred simultaneously. While it may produce satisfactory sequences which regenerates the desired shape, it is unclear how the successive pillars interact with one another. In this section, we propose a deep learning architecture which predicts the intermediate shape between two images and learns a sequence of causal actions contributing to desired shape transformation. The architecture can be easily implemented in applications such as:
1. Learning to transform the belief space for robotic path planning. A robot may be chasing a mobile target while maintaining a posterior (that is transformed into a visual representation analogous to the flow shape) corresponding to the location of the target. If we would like to specifically maximize posterior in a certain region, the corresponding problem would be: “What is the best course of action that should be taken by the robot in succession?".

2. Learning the material processing pathways to obtain the desired microstructures. In materials processing, processing the materials by altering the properties in succession will alter the morphology (microstructures) of the material. The equivalent inverse problem would be: “If we want the material to achieve a specific morphology, what are the processing steps that should be taken?".

3. Learning a sequence of manufacturing steps in additive manufacturing, with fast-design being the main advantage.

While CNN-SMC is capable of predicting a large number of different sequences in a total time of just seconds, a drawback is that the sequence length is constrained because a new model needs to be trained to generate a sequence with different lengths. Furthermore, the sequence which deforms into the target flow shape is predicted in a joint manner and do not provide sufficient insight on the interplay between pillars causing the deformation.

Recurrent neural network (RNN)-like architectures [90] are, although scalable, deemed unsuitable because the elements in the output vector in our problem are generally independent of each other, unlike words in a sentence. In this application, a pillar can be independently added to the flow channel only after the flow shape resulted from the previous pillar stabilizes, i.e. no longer changes. This way, we guarantee that there are no interactions among pillars (hence the independence). However, all these pillars contribute to the change in the overall flow shape (thus they are causal to the final shape). In the context of RNN outputs, e.g. for the case of next-word prediction, oftentimes the output vectors are not independent—one word will have a high probability to be the next word of a certain word. In our context, the choice of the next pillar will not depend on the choice of the previous pillar. Rather, we look at the now by making use of the given generated shape so far regardless of the history.
A related concept is the spatial transformer network [56] where the localization network outputs geometrical transformation parameters; however we desire an exact class attributed to an arbitrary transformation function. In this work, we predict the pillar sequence one pillar at a time without disregarding causality, such that the produced sequences deform the flow into one that resembles more closely to the desired shape.

### 3.5.1 Proposed architectures

In this section, we present two core architectures and the integrated framework which fuses these two architectures.

#### 3.5.1.1 Pillar prediction network (PPN)

To predict the sequence of pillars that results in the desired flow shape, this chapter introduces the notion of *transformation learning*. Fig. 3.9 shows the learning approach by supplying juxtaposed flow shapes, one before deformation, and one after, into a CNN that extracts relevant features and predicts the class of the pillar causing the deformation. In the training data generation procedure, the input is comprised of three parts: (a) the pre-deformed flow shape, which is produced from a randomly generated sequence of varying length with values up to the number of classes of a single pillar; (b) a padding to prevent the convolutional kernels to pick up interfering features from the juxtaposed images; and (c) the post-deformed shape

![Figure 3.9: The PPN addresses the question: ‘Given a pair of pre- and post-deformed shape, what is the identity of the pillar causing the deformation?’](image)

Figure 3.9: The PPN addresses the question: ‘Given a pair of pre- and post-deformed shape, what is the identity of the pillar causing the deformation?’

To predict the sequence of pillars that results in the desired flow shape, this chapter introduces the notion of *transformation learning*. Fig. 3.9 shows the learning approach by supplying juxtaposed flow shapes, one before deformation, and one after, into a CNN that extracts relevant features and predicts the class of the pillar causing the deformation. In the training data generation procedure, the input is comprised of three parts: (a) the pre-deformed flow shape, which is produced from a randomly generated sequence of varying length with values up to the number of classes of a single pillar; (b) a padding to prevent the convolutional kernels to pick up interfering features from the juxtaposed images; and (c) the post-deformed shape...
produced from adding a random pillar index to the previous sequence. This newly-added pillar index will become the label to train the CNN. Essentially, the pillar prediction network, as the name suggests, predicts the index of the pillar (which describes its position and diameter in the flow channel) given a pre- and post-deformed shape.

When actually inferencing, the right portion (part c) of the input image is replaced by the final target shape. The input image is fed into the CNN to obtain a pillar index, which is added to an initially empty pillar sequence. Because the forward model (i.e. going from sequence to flow shape) is not at all computationally expensive and takes merely milliseconds, this pillar sequence is used to regenerate the current shape which replaces the left portion (part a) of the input, while the right portion (part c, the target) remains the unchanged. The input enters the CNN again to obtain a second pillar index, and is subsequently added to the previously obtained pillar sequence. At this point, we have a sequence of length 2, where the sequence will then again used to regenerate a new current shape. The process is repeated until the current shape matches the target shape, or until a user-defined stopping criterion is met.

![Inferencing with CNN with isolated channels](image)

Figure 3.10: PPN-C treats the pre- and post-deformed shapes as separate channels.

An alternative PPN is to feed the pre- and post-deformed shapes into the CNN separately with isolated channels before merging them together in the fully connected layer 3.10. This architecture will be referred to as PPN-C (where the C denotes channels and signifies that the pre- and post-deformed shapes are treated as different channels). In other words, each channel has different sets of filters and the relationship between the two sets will fuse together at the fully connected layer.
However, this method only works well for simpler target shapes. For more complex flow shapes (e.g. shapes with many sharp angles, jagged edges, swirls and curls; see Fig. 3.15 sample 20A for example), the transformation path may be highly nonlinear—the current shape may never converge to the final desired shape. Furthermore, the training data covers a vanishingly small fraction of the design space with coverage shrinking exponentially as the sequence length increases (i.e., an \( n_p \) sequence will result in \( 32^{n_p} \) different combinations), so it is necessary to learn the transformations in a meaningful way. To help alleviate this issue, we will introduce the Intermediate Transformation Network (ITN) in the next subsection.

**PPN parameters:** The input image into the PPN is comprised of two \( 12 \times 100 \) px flow shape image with a \( 5 \times 100 \) px padding in between, resulting in a final dimension of \( 29 \times 100 \) px. For PPN-C, the two inputs are treated as separate channels without introducing padding. The model contains two convolutional layers with 40 and 100 kernels respectively (sizes of \( 5 \times 5 \) and \( 3 \times 3 \) px), each followed by a \( 2 \times 2 \) maxpooling layer. The fully connected layer has 500 hidden units. Training is done with 250,240 training examples and 60,000 validation examples in minibatches of 50 with a learning rate of 0.01. The training procedure employs the early stopping algorithm where training stops when validation error ceases to decrease.

### 3.5.1.2 Intermediate transformation network (ITN)

**Figure 3.11:** The ITN addresses the question: ‘Given the final shape, what is the possible shape that lies in the middle of the nonlinear deformation pathway?’

The ITN attempts to construct a flow shape that bridges between the purely undeformed flow shape (i.e. shape generated with an empty sequence) and the final desired flow shape in...
the nonlinear transformation path (See Fig. 3.11(a)). We used a deep autoencoder to extract hierarchical features from the desired final shape, and outputs an approximated bridging shape. To generate the training data for this network, the input image is generated with a random pillar sequence with a varying length. The corresponding bridging shape is generated by truncating the same pillar sequence by half, thus the shape lies in the middle of the transformation pathway (Fig. 3.11(b)). Formally, if a sequence has length \( n \), it is truncated to \( (n + 1)/2 \) if \( n \) is odd, and to \( n/2 \) if \( n \) is even. This pair of images is used to train a deep autoencoder, where the mean squared error (MSE) between the model outputs and the desired bridging shape is backpropagated to finetune the weights and biases of the deep autoencoder.

During inference, the edges in the outputs of the ITN may appear blur because the bridging shape is only an approximation (Fig. 3.11(c)). To obtain pure black and white images, we threshold the pixel values where pixel intensities larger or equal to 0.5 are set to 1, 0 otherwise. The ITN may be extended to obtain several waypoints instead of only the midpoint. Doing so will allow a smoother transformation pathway, but will require some changes in the training data generation procedure. We call this the recursive ITN (rITN) and leave this as a future work. Some may comment that ITN is somewhat similar to RNN. We note that for this problem (along with the example applications discussed in section 3.5) requires transforming the hidden states into the physical domain using a complex forward function before making the next prediction. Unlike RNNs, ITNs do not solely use the latent output directly as the input in the next time step. Due to the forward function, we need to constantly transform the output back into the physical domain and re-evaluate at every step. One may also argue that ITN is somewhat similar to RNN in terms of memory, but by the same argument, the difference in domain is the key. If only the latent domain is used for every step, the final sequence output may not necessarily reflect the true transformation in the physical domain.

**ITN parameters:** The autoencoder has 3 layers of 500 hidden units each and accepts flow shape images of \( 12 \times 100 \) px. Training was done on 500,000 training examples and 20,000 validation examples in minibatch of size 1,000 with a learning rate of 0.01. Training is also done using the early-stopping algorithm.
3.5.1.3 The integrated pipeline (PPN+ITN)

With the roles of the PPN and ITN clearly described, we can now integrate both networks to form an integrated pipeline. A schematic is shown in Fig. 3.12. At the very beginning, the ITN guesses a candidate for the bridging flow shape, which is then considered as the temporary target shape, and is concatenated with the undeformed shape placed to its left with a padding. The concatenated input is supplied into the PPN to predict the first pillar causing the deformation, and then added to the sequence which was initially empty, hence resulting in a sequence of length 1. The current shape is regenerated with the updated 1-pillar sequence and replaces the left portion of the input image for the next PPN iteration to obtain the second pillar index. The process is repeated as ‘Stage A’ (with the bridging shape as a temporary target) until the current shape is sufficiently similar to the bridging shape, or until an iteration limit is reached. Then, the right portion of the input image (which was the bridging shape) is replaced with the final desired shape as the target. This process is continued as ‘Stage B’ which undergoes the same process as ‘Stage A’, except the target shape is now the final desired shape. After each iteration, the predicted pillar index is added to the sequence until the reconstructed flow shape matches the desired shape or a stopping criterion is achieved. The resulting sequence will vary in length for different desired shapes.
3.5.2 Results and discussions

In this section we first present the evaluation metrics used in the study, then show the results comparing CNN-SMC against our method using PPN and the PPN+ITN hybrid architecture.

To quantify the effectiveness of both approaches, we evaluated the PMR on 20 target flow shapes and computed appropriate statistics. Since shape transformation is an essential aspect in this problem formulation, the structural similarity index (SSIM) \([147]\) is used as a supplementary metric to compare how structurally similar are the regenerated flow shape images (from predicted sequence) to the target flow shape. SSIM explores the change in image structure and incorporates pixel inter-dependencies. SSIM is expressed as:

\[
SSIM(x, y) = \frac{(2\mu_x\mu_y + c_1)(2\sigma_{xy} + c_2)}{(\mu_x^2 + \mu_y^2 + c_1)(\sigma_x^2 + \sigma_y^2 + c_2)}
\]

where \(\mu_x\) is the average of window \(x\), \(\mu_y\) is the average of window \(y\), \(\sigma_x^2\) is the variance of \(x\), \(\sigma_y^2\) is the variance of \(y\), \(\sigma_{xy}\) is the covariance of \(x\) and \(y\), \(c_1 = (k_1L)^2\) and \(c_2 = (k_2L)^2\) are two variables to stabilize the division with weak denominator with \(k_1 = 0.01\) and \(k_2 = 0.03\) by default, and \(L\) is the dynamic range of pixel values.

3.5.2.1 Predicting sequences with PPN and PPN+ITN

In all of our tests, the target flow shape is generated from a 10-pillar sequence which is sufficiently complex. Fig. 3.13 shows four example target shapes with the performance using only PPN (without bridging) and PPN+ITN (with bridging). In most cases, the PPN-only formulation produces pillar sequences resulting in flow shapes that do not resemble the target shape as close as using PPN+ITN. This can be clearly seen for cases C and D in Fig. 3.13. On the other hand, by using the bridging shape as a temporary target, the prediction performance saw great improvements. In addition, we see that most shapes are able to converge to both the bridging shape as well as the target in the PPN+ITN formulation. In realistic applications, the sequence may be stored in memory and post-processed to remove the redundant pillars, thus producing a shorter sequence which may increase financial savings during the manufacturing process.
### 3.5.2.2 Comparison of CNN+SMC, PPN, PPN-C, and PPN+ITN

20 sample-wise comparison on the performance of CNN+SMC, and our methods PPN and PPN+ITN are shown in Fig. 3.14 and tabulated in Table 3.4. The reconstructed flow shapes from the predicted sequences are shown in Fig. 3.15. In both Fig. 3.14(a) and Fig. 3.14(b), the PMR and SSIM for PPN, PPN-C, and PPN+ITN are clearly higher than the CNN+SMC approach. We observe that for some target flow shapes, the predicted sequence for CNN+SMC may result in an entirely dissimilar shape (e.g. sample 2, 19, 20). In some cases (e.g. samples 14, 15, 16) PPN fared better than the hybrid PPN+ITN model. However, PPN+ITN is consistently more superior in terms of both PMR and SSIM than the PPN-only architecture (i.e. PPN and PPN-C). This shows that having a bridging shape generally helps in producing sequences that generate complex flow shapes. The inferior performance of CNN+SMC is due the model being constrained to always generate a sequence with 10 pillars, thus the resultant flow shapes often become overcomplicated.

#### Figure 3.13: Four examples of sequence prediction using (i) PPN-only without bridging and (ii) PPN+ITN with bridging. By predicting a bridging shape, the resulting predicted sequence is able to reconstruct flow shapes that are more similar to the target shape. Each frame shows the deformation on the flow shape with each additional predicted pillar added into the sequence.
Additionally, treating the input as separate channels results in lower performance while all other model parameters are kept equal. This suggests that having a single filter that learns features from both input images simultaneously is more effective in feature extraction compared to learning the features in isolation before merging at the fully connected layer.

A clear advantage of using both PPN and ITN together is that the model does not need to be retrained for variable sequence lengths, unlike in the CNN-SMC model where the number of pillars in the output sequence is constrained. This method is highly scalable, and has an enormous room for extension into sculpting more highly complex flow shapes.

3.6 Conclusions and Future Work

This paper proposes a deep learning based approach to solve complex design exploration problems, specifically design of microfluidic channels for flow sculpting. We have demonstrated that DL based tools can achieve the required design accuracy. The state-of-the-art GA based method performs slightly better quantitatively, but it remains a less desirable solution due very long execution time. DL based methods can expedite the design process by more than 600 times and present a real-time design alternative. A quasi-random training data generation process was developed based on a metric on the output space. Such a process can be used for learning
Table 3.4: PMR and SSIM of regenerated flow shapes using different enhancement methods. The numbers reported are in the format of [PMR/SSIM]. Bolded numbers correspond to the method with the highest PMR and SSIM. Asterisk (*) denotes our architecture presented in this paper.

<table>
<thead>
<tr>
<th>Target Flow Shapes</th>
<th>CNN+SMC</th>
<th>PPN*</th>
<th>PPN-C*</th>
<th>PPN+ITN*</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test Shape 1</td>
<td>0.5600 / 0.1829</td>
<td>0.8317 / 0.4349</td>
<td><strong>0.8867 / 0.5450</strong></td>
<td>0.8475 / 0.4521</td>
</tr>
<tr>
<td>Test Shape 2</td>
<td>0.5133 / 0.0578</td>
<td><strong>0.9375 / 0.7160</strong></td>
<td>0.7408 / 0.2620</td>
<td>0.9117 / 0.5899</td>
</tr>
<tr>
<td>Test Shape 3</td>
<td>0.9367 / 0.6622</td>
<td>0.9342 / 0.6437</td>
<td>0.9400 / 0.6766</td>
<td><strong>0.9558 / 0.7481</strong></td>
</tr>
<tr>
<td>Test Shape 4</td>
<td>0.5883 / 0.1813</td>
<td>0.8833 / 0.5303</td>
<td>0.8550 / 0.5315</td>
<td><strong>0.8950 / 0.6082</strong></td>
</tr>
<tr>
<td>Test Shape 5</td>
<td>0.8275 / 0.4539</td>
<td>0.9650 / 0.8108</td>
<td>0.9525 / 0.7710</td>
<td><strong>0.9725 / 0.8407</strong></td>
</tr>
<tr>
<td>Test Shape 6</td>
<td>0.6992 / 0.2705</td>
<td>0.9208 / 0.6765</td>
<td>0.9283 / 0.6959</td>
<td><strong>0.9358 / 0.7346</strong></td>
</tr>
<tr>
<td>Test Shape 7</td>
<td>0.6300 / 0.1412</td>
<td><strong>0.9567 / 0.7895</strong></td>
<td>0.9500 / 0.7760</td>
<td>0.9558 / <strong>0.8167</strong></td>
</tr>
<tr>
<td>Test Shape 8</td>
<td>0.6117 / 0.2555</td>
<td>0.8125 / 0.4687</td>
<td>0.8758 / 0.5802</td>
<td><strong>0.9075 / 0.6330</strong></td>
</tr>
<tr>
<td>Test Shape 9</td>
<td>0.7767 / 0.3743</td>
<td>0.9117 / 0.5776</td>
<td>0.9108 / 0.5731</td>
<td><strong>0.9292 / 0.6105</strong></td>
</tr>
<tr>
<td>Test Shape 10</td>
<td>0.7883 / 0.4407</td>
<td>0.9500 / 0.7745</td>
<td>0.9508 / 0.7646</td>
<td><strong>0.9625 / 0.8299</strong></td>
</tr>
<tr>
<td>Test Shape 11</td>
<td>0.5892 / 0.2046</td>
<td>0.8608 / 0.4137</td>
<td>0.8383 / 0.5346</td>
<td><strong>0.9275 / 0.6511</strong></td>
</tr>
<tr>
<td>Test Shape 12</td>
<td>0.6058 / 0.1430</td>
<td>0.9025 / <strong>0.5527</strong></td>
<td>0.8483 / 0.4760</td>
<td><strong>0.8908 / 0.5314</strong></td>
</tr>
<tr>
<td>Test Shape 13</td>
<td>0.7200 / 0.2126</td>
<td><strong>0.9442 / 0.7458</strong></td>
<td>0.9400 / 0.7391</td>
<td>0.9433 / <strong>0.7233</strong></td>
</tr>
<tr>
<td>Test Shape 14</td>
<td>0.5983 / 0.2229</td>
<td>0.9433 / 0.6804</td>
<td><strong>0.9450 / 0.6804</strong></td>
<td>0.8850 / 0.5466</td>
</tr>
<tr>
<td>Test Shape 15</td>
<td>0.6808 / 0.3412</td>
<td>0.8933 / 0.5195</td>
<td><strong>0.9442 / 0.6980</strong></td>
<td>0.8583 / 0.4332</td>
</tr>
<tr>
<td>Test Shape 16</td>
<td>0.5508 / 0.1135</td>
<td><strong>0.9433 / 0.6919</strong></td>
<td>0.8100 / 0.3533</td>
<td>0.9200 / 0.5813</td>
</tr>
<tr>
<td>Test Shape 17</td>
<td>0.7792 / 0.4134</td>
<td>0.9033 / 0.5882</td>
<td>0.9408 / 0.7010</td>
<td><strong>0.9617 / 0.7655</strong></td>
</tr>
<tr>
<td>Test Shape 18</td>
<td>0.6133 / 0.2677</td>
<td>0.9075 / 0.6112</td>
<td>0.9058 / 0.5567</td>
<td><strong>0.9267 / 0.6416</strong></td>
</tr>
<tr>
<td>Test Shape 19</td>
<td>0.6075 / 0.0464</td>
<td>0.9367 / 0.7540</td>
<td>0.8967 / 0.6324</td>
<td><strong>0.9567 / 0.8044</strong></td>
</tr>
<tr>
<td>Test Shape 20</td>
<td>0.5300 / 0.0134</td>
<td>0.8983 / 0.6039</td>
<td><strong>0.9325 / 0.7051</strong></td>
<td>0.9183 / 0.6753</td>
</tr>
<tr>
<td>Average</td>
<td>0.6603 / 0.2500</td>
<td>0.9118 / 0.6292</td>
<td>0.8996 / 0.6126</td>
<td><strong>0.9231 / 0.6609</strong></td>
</tr>
</tbody>
</table>
Figure 3.15: 20 test shapes with reconstructed flow shapes generated from sequences predicted using different methods. Column A is the target shape, B for the reconstruction for CNN+SMC, C for PPN, D for PPN-C, and E for PPN+ITN.

features from simulation and experimental data in many physical problems in general. The training data generation and hierarchical feature extraction processes together prove to be quite useful for a scalable design tool. Finally, feature visualization is performed to investigate flow pattern aspects at various scales.

We also proposed a deep architecture to learn a sequence of actions that carries out the desired transformation over the input which has great implications on the innovation of manufacturing processes, material sciences, biomedical applications, decision planning, and many more. The creative integration of DL based tools can tackle the inverse fluid problem and achieve the required design accuracy while expediting the design process.

Current efforts are primarily focusing on optimizing the tool-chain (e.g. rITN), defining and quantifying the complexity of a flow shape, as well as tailoring for specific application areas such as manufacturing and biology.
CHAPTER 4. EARLY DETECTION OF COMBUSTION INSTABILITY USING HI-SPEED FLAME VIDEOS

4.1 Motivation

Strict emission regulation has initiated a paradigm shift in the nominal operating conditions of gas turbine engines. Consequently, the technology of gas turbine engines has gradually adapted to low equivalence ratio combustion to suppress emissions of nitrogen oxides (NOx), instead of combustion at near-stoichiometric conditions. Ultra-lean premixed and pre-vaporized combustors, while being environment-friendly, is susceptible to combustion instabilities that are typically characterized by pressure waves with sharp tones and high amplitudes. The complexity of such an instability problem accrues from the mutual interactions among the unsteady heat release rate, flow fields and acoustics, which outline the general features of combustion instability [78, 88, 32]. Combustion instabilities have many detrimental effects on flight-propulsion dynamics and structural integrity of gas turbine engines and are characterized by high-amplitude flame oscillations at discrete frequencies. These frequencies typically represent the natural duct/resonator acoustic modes. Combustion instability, in its most basic form arises when there is a positive coupling between the heat release rate oscillations and the pressure oscillations, provided this driving force is higher than the damping present in the system. The mechanisms of pressure-heat release rate coupling are system dependent and thus, the problem of combustion instability becomes very system specific.

The underlying principle of heat release rate oscillations, that drives the pressure oscillations—which result in velocity oscillations and in turn modulate heat release rate oscillations—all in a turbulent background in case of actual gas turbine combustors pose significant complexities in determining the mechanisms of combustion instability. Crocco [88] modeled unsteady heat release
rate as a function of unsteady velocity to determine stability of a ducted zero-mean flow flame. Subsequently, a whole class of reduced order modeling-Flame Transfer/Describing functions were theoretically and experimentally [99, 98, 12] formulated to understand the stability of the system by means of solving the dispersion relation. In addition, flame oscillation saturation mechanisms were also experimentally diagnosed which in addition to experiments based on turbulent non-reacting and reacting flows led to the universal feature of combustion instability- heat release rate oscillations driven by coherent structures.

Coherent structures are fluid mechanical structures associated with coherent phase of vorticity, high levels of vorticity among other definitions [55]. These structures, whose generation mechanisms vary system wise, cause large scale velocity oscillations and overall flame shape oscillations by curling and stretching. These structures can be caused to shed/generated at the duct acoustic modes when the forcing (pressure) amplitudes are high. The interesting case of the natural shedding frequency of these structures, causing acoustic oscillations, has been observed by Chakravarthy et al. [23].

Recently, a swirl combustor has been characterized and a wide range of experiments relating swirl flows and coherent structures associated with swirl flows has been reported [138, 101]. The presence of Precessing vortex core as the dominant coherent structure has been reported and non linear interactions between heat release rate oscillations and PVC as the cause of superposed frequencies in time series data has also been reported [92]. Much of the literature is dedicated to detection and correlation of these coherent structures to heat release rate and unsteady pressure. The popular methods resorted for detection of coherent structures are proper orthogonal decomposition (POD) [17] and dynamic mode decomposition (DMD) [128], which use tools from spectral theory to derive spatial coherent structure modes. DMD has been used to estimate the growth rates and frequencies from experimental data and thus offered to perform stability analysis on experimental data.

This study proposes a data-driven hierarchical framework for early detection of thermo-acoustic instability from hi-speed greyscale images. In the lower layer (lower feature hierarchy), large volume of hi-speed sequential images are used to train a deep neural network model that extracts hierarchical features from the training data [51] through the use of multiple layers of
latent variables. An unsupervised pre-training approach with deep-belief networks (DBN) [49] is used in particular to automatically learn the coherent structures while reducing the dimension of the images for temporal modeling at the top layer (higher feature hierarchy) [37]. Symbolic time series analysis (STSA) [110], a fast probabilistic graphical model is placed at the top layer to extract temporal feature from the output of deep learning model. The concept of STSA has been used for anomaly detection in physical systems as reported in [110, 109, 119]. Recently, STSA is applied on pressure and chemiluminescence time series for early detection of Lean-blow out [95, 123] and thermo-acoustic instability [108].

Additionally, we present a sensor-fusion-based approach for early detection of instability phenomena in combustors. Sensors of different modalities (e.g., hi-speed grey-scale images and pressure time-series data) have been used to observe the same physical process that can capture the expected outcome of one modality based on the state of the other modality. To develop cross-models for multimodal sensor fusion, first a DBN [49] is used to process the hi-speed grey-scale images for dimensionality reduction and automatically learning the features from the combustion images. Then, the cross-models of the features from images and pressure time-series data are represented as a $\times D$-Markov machine [134, 125, 118] that is capable of capturing the behavior of a symbolic process conditioned on another symbolic process. These $\times D$-Markov machine models are then used to detect and predict combustion instabilities. The proposed approach offers the benefit that data from two different sensor modalities can be used to arrive at more robust and accurate prediction of the underlying physical process, and its efficacy has been validated using experimental data of combustion from a laboratory-scale swirl combustor apparatus [108]; in particular, hi-speed grey-scale images and pressure time-series data have been used for detection and prediction of combustion instabilities in a laboratory-scale swirl combustor apparatus [108].

From the above perspectives, the major contributions of the study are delineated below.

- A novel data-driven framework, with DBN or CNN at lower layer and STSA at upper layer, is proposed for early detection of thermo-acoustic instability from hi-speed images.
Figure 4.1: (a) Schematic of the experimental setup. 1 - settling chamber, 2 - inlet duct, 3 - IOAM, 4 - test section, 5 - big extension duct, 6 - small extension ducts, 7 - pressure transducers, $X_s$ - swirler location measured downstream from settling chamber exit, $X_p$ - transducer port location measured downstream from settling chamber exit, $X_i$ - fuel injection location measured upstream from swirler exit, (b) Swirler assembly used in the combustor

- In the above framework, the DBN and CNN layers extract meaningful shape-features to represent the coherent structures in the flame images. This phenomenon enables STSA at the temporal modeling layer to enhance the class separability between stable and unstable modes of combustion, which implies higher precision for early detection of the onset of combustion instability.

- The proposed theory and the associated algorithms have been experimentally validated at multiple operating conditions in a swirl-stabilized combustor by characterizing the stable and unstable states of combustion.

- Training and testing of the proposed framework have been performed on different operating conditions (e.g., Reynolds number ($Re$), fuel flow rate, and air-fuel premixing level) of the combustion process to test the transferability of the approach. Performance of the proposed framework (DBN+STSA and CNN+STSA) have been evaluated by comparison with that of a framework, where DBN or CNN is replaced by another extensively used dimensionality reduction tool, principal component analysis (PCA) [18].

4.2 Experimental Setup

The swirl combustor test bed used in this study has a swirler of diameter 30 mm with 60 degree vane angles, thus yielding a geometric swirl number of 1.28. Air to the combustor is
fed through a settling chamber of diameter 280 mm with a sudden contraction leading to a square cross section of side 60 mm. This provides an area ratio of around 17, which thus acts as an acoustically open condition at the contraction. A mesh and honeycomb are mounted in immediate downstream of the contraction to provide uniform flow to the swirler. The combustor, shown in Fig. 4.1(a) consists of an inlet section of length 200 mm, an inlet optical access module (IOAM) of length 100 mm to provide optical access to the fuel tube, a primary combustion chamber of length 370 mm, and secondary duct of the same length. Extension ducts of the same cross section are added to provide length flexibility. The overall length of the constant area ducts was chosen to be 1340 mm.

Fuel injection is done by injecting it coaxially with the air in a fuel injection tube with slots on the surface as shown in Fig. 4.1(b). The fuel injection tube is coaxial to a mixing tube which has the same diameter as that of the swirler. The bypass air that does not enter the mixing tube passes through slots on the swirl plate. The slots on the fuel injection tube are drilled at designated distance upstream of the swirler. The larger this distance, more fuel mixes with the primary air in the mixing tube thus leading to more premixedness. Two upstream distances of $X_1 = 90\, mm$ and $X_2 = 120\, mm$ were chosen for this work. The upstream distance of 120 mm provides for full premixing of the fuel with the air thus henceforth, it will be referred to as the premixed case. The 90 mm upstream injection case causes partial premixing of the fuel with air and thus will be referred to as the partially premixed case. The images were acquired at 3 kHz using Photron High speed star with a spatial resolution of $1024 \times 1024$ pixels. The data acquisition was triggered simultaneously using NI card and taken for a duration of 3s yielding in a sequence of 9,000 images for every operating condition.

Two inlet Reynolds numbers (Re), based on the swirler diameter were chosen, the lower Re having stable combustion behavior and higher Re having exhibiting unstable behavior. The Re’s were chosen to be 7,971 and the higher Re being 15,942 for a fuel flow rate (FFR) of 0.495 g/s. Another protocol followed was keeping the inlet Re constant at 10,628 and having two different fuel flow rates. The higher FFRs exhibited stable combustion, whereas the leaner configuration was unstable. The two FFRs were chosen to be 0.66 g/s and 0.308 g/s. These corresponded to equivalence ratios of 0.955 and 0.445 respectively. Besides these conditions, 3 seconds of images
Table 4.1: Description of operating conditions along with respective ground truth (stable or unstable) for hi-speed image data collection. 3s of greyscale image sequence at 3kHz is collected for each condition.

<table>
<thead>
<tr>
<th>Premixing</th>
<th>FFR (g/s)</th>
<th>Re</th>
<th>Ground truth</th>
</tr>
</thead>
<tbody>
<tr>
<td>Partial</td>
<td>0.495</td>
<td>7,971</td>
<td>Stable</td>
</tr>
<tr>
<td></td>
<td></td>
<td>15,942</td>
<td>Unstable</td>
</tr>
<tr>
<td>(X₁ = 90mm)</td>
<td>0.308</td>
<td>10,628</td>
<td>Unstable</td>
</tr>
<tr>
<td></td>
<td>0.66</td>
<td></td>
<td>Stable</td>
</tr>
<tr>
<td>Full</td>
<td>0.495</td>
<td>7,971</td>
<td>Stable</td>
</tr>
<tr>
<td></td>
<td></td>
<td>15,942</td>
<td>Unstable</td>
</tr>
<tr>
<td>(X₂ = 120mm)</td>
<td>0.308</td>
<td>10,628</td>
<td>Unstable</td>
</tr>
<tr>
<td></td>
<td>0.66</td>
<td></td>
<td>Stable</td>
</tr>
<tr>
<td></td>
<td>0.083</td>
<td>1,771</td>
<td>Relatively stable</td>
</tr>
</tbody>
</table>

Figure 4.2: Top: greyscale images at $Re = 7,971$ and full premixing for a fuel flow rate of 0.495 g/s, bottom: greyscale images at $Re = 15,942$ and full premixing for a fuel flow rate of 0.495 g/s are also collected for $Re = 1,771$ and $FFR = 0.083$ at relatively stable state of combustion. The details of the operating conditions along with their ground truth (e.g., stable or unstable) are presented in table 4.1.

Fig. 4.2 presents sequences of images of dimension $392 \times 1000$ pixels for both stable ($Re = 7,971$, $FFR = 0.495g/s$ and full premixing) and unstable ($Re = 15,942$, $FFR = 0.495g/s$ and full premixing) states. The flame inlet is on the right side of each image and the flame flows downstream to the left. It can be observed that the flame does not have any prominent coherent structure when the combustion is stable. While the combustion is unstable, vortex shedding along the flow is observed. Bottom of the Fig. 4.2 shows formation of mushroom-shaped vortex at $t = 0, 0.001s$ and the shedding of that towards downstream from $t = 0.002s$ to $t = 0.004s$. 
4.3 Deep learning and symbolic time series analysis: an integrated framework

This section describes the proposed architecture for early detection of thermo-acoustic instability in a combustor via analyzing a sequence of hi-speed images. Fig. 4.3 presents the schematics of the framework where a DBN or CNN is stacked with symbolic time series analysis (STSA). In the training phase, images (or a segment of the images) from both stable and unstable states for various operating conditions are used as the visible layer $V$ of a deep network. In this work, an unsupervised pre-training step is emphasized more for capturing the coherent structures in flame images at unstable state. The vector of activation probabilities of the hidden units at the topmost hidden layer is used as input to the STSA module.

While testing, sequence of images are passed through the learned neural network and a time series of $l_2$ norm (equivalent to signal energy) of the activation probability vectors is obtained. In STSA module, the time-series is symbolized via partitioning the signal space and a symbol sequence is created as shown in the Fig. 4.3. A probabilistic finite state automata (PFSA) [110] is constructed from the symbol sequence, which models the transition from one state to another as state transition matrix. State transition matrix is the extracted feature which represents the sequence of images, essentially capturing the temporal evolution of coherent structures in the flame. DBN and STSA are explained in detail later in this section.

4.4 Background on symbolic time series analysis (STSA)

STSA [30] deals with discretization of dynamical systems in both space and time. The notion of STSA has led to the development of a (nonlinear) data-driven feature extraction tool for dynamical systems. Rao et al. [109] and Bahrampour et al. [9] have shown that the performance of this PFSA-based tool as a feature extractor for statistical pattern recognition is comparable (and often superior) to that of other existing techniques (e.g., Bayesian filters, Artificial Neural Networks, and Principal Component Analysis [18]). The trajectory of the dynamical system is partitioned into finitely many mutually exclusive and exhaustive cells for symbolization, where each cell corresponds to a single symbol belonging to a (finite) alphabet $\Sigma$. 
Figure 4.3: Framework for early detection of combustion instability from hi-speed flame images via semantic feature extraction using deep belief network (DBN) followed by symbolic time series analysis (STSA)
There are different types of partitioning tools, such as maximum entropy partitioning (MEP), uniform partitioning (UP) [107] and maximally bijective partitioning [127]. This paper has adopted MEP for symbolization of time series, which maximizes the entropy of the generated symbols by putting (approximately) equal number of data points in each partition cell. The next step is to construct probabilistic finite state automata (PFSA) from the symbol strings to encode the embedded statistical information. PFSA is a 4-tuple $K = (\Sigma, Q, \delta, \pi)$ which consists of a finite set of states ($Q$) interconnected by transitions [130], where each transition corresponds to a symbol in the finite alphabet ($\Sigma$). At each step, the automaton moves from one state to another (including self loops) via transition maps ($\delta : Q \times \Sigma \rightarrow Q$) according to probability morph function ($\tilde{\pi} : Q \times \Sigma \rightarrow [0,1]$), and thus generates a corresponding block of symbols so that the probability distributions over the set of all possible strings defined over the alphabet are represented in the space of PFSA.

### 4.4.1 Generalized D-Markov machine [123]

D-Markov machine is a model of probabilistic languages based on the algebraic structure of PFSA. In D-Markov machines, the future symbol is causally dependent on the (most recently generated) finite set of (at most) $D$ symbols, where $D$ is a positive integer. The underlying FSA in the PFSA of D-Markov machines are deterministic. The complexity of a D-Markov machine is reflected by the entropy rate which also represents its overall capability of prediction. A D-Markov machine and its entropy rate are formally defined as:

**Definition 4.4.1 (D-Markov)** A D-Markov machine is a statistically stationary stochastic process $S = \cdots s_{-1}s_0s_1\cdots$ (modeled by a PFSA in which each state is represented by a finite history of at most $D$ symbols), where the probability of occurrence of a new symbol depends only on the last $D$ symbols, i.e.,

$$P[s_n \mid \cdots s_{n-D} \cdots s_{n-1}] = P[s_n \mid s_{n-D} \cdots s_{n-1}] \tag{4.1}$$

$D$ is called the depth. $Q$ is the finite set of states with cardinality $|Q| \leq |\Sigma|^D$, i.e., the states are represented by equivalence classes of symbol strings of maximum length $D$, where each symbol belongs to the alphabet $\Sigma$. $\delta : Q \times \Sigma \rightarrow Q$ is the state transition function that satisfies the
following condition: if \(|Q| = |\Sigma|^D\), then there exist \(\alpha, \beta \in \Sigma\) and \(x \in \Sigma^*\) such that \(\delta(\alpha x, \beta) = x\beta\) and \(\alpha x, x\beta \in Q\).

**Definition 4.4.2 (\(D\)-Markov Entropy Rate)** The \(D\)-Markov entropy rate of a PFSA \((\Sigma, Q, \delta, \pi)\) is defined in terms of the conditional entropy as:

\[
H(\Sigma|Q) \triangleq \sum_{q \in Q} P(q)H(\Sigma|q) = -\sum_{q \in Q} \sum_{\sigma \in \Sigma} P(q) P(\sigma|q) \log P(\sigma|q)
\]

where \(P(q)\) is the probability of a PFSA state \(q \in Q\) and \(P(\sigma|q)\) is the conditional probability of a symbol \(\sigma \in \Sigma\) given that a PFSA state \(q \in Q\) is observed.

**4.4.2 Construction of a \(D\)-Markov machine [94]**

The underlying procedure for construction of a \(D\)-Markov machine from a symbol sequence consists of two major steps: state splitting and state merging [94, 124]. In general, state splitting increases the number of states to achieve more precision in representing the information content of the dynamical system. State merging reduces the number of states in the \(D\)-Markov machine by merging those states that have similar statistical behavior. Thus, a combination of state splitting and state merging leads to the final form of the generalized \(D\)-Markov machine as described below.

**State Splitting:** The number of states of a \(D\)-Markov machine of depth \(D\) is bounded above by \(|\Sigma|^D\), where \(|\Sigma|\) is the cardinality of the alphabet \(\Sigma\). As this relation is exponential in nature, the number of states rapidly increases as \(D\) is increased. However, from the perspective of modeling a symbol sequence, some states may be more important than others in terms of their embedded information contents. Therefore, it is advantageous to have a set of states that correspond to symbol blocks of different lengths. This is accomplished by starting off with the simplest set of states (i.e., \(Q = \Sigma\) for \(D = 1\)) and subsequently splitting the current state that results in the largest decrease of the \(D\)-Markov entropy rate. The process of splitting a state \(q \in Q\) is executed by replacing the symbol block \(q\) by its branches as described by the set \(\{\sigma q : \sigma \in \Sigma\}\) of words. Maximum reduction of the entropy rate is the governing criterion for selecting the state to split. In addition, the generated set of states must satisfy the self-consistency criterion, which only permits a unique transition to emanate from a state for a given
symbol. If $\delta(q, \sigma)$ is not unique for each $\sigma \in \Sigma$, then the state $q$ is split further. The process of state splitting is terminated by either the threshold parameter $\eta_{spl}$ on the rate of decrease of entropy rate or a maximal number of states $N_{max}$. For construction of PFSA, each element $\pi(\sigma, q)$ of the morph matrix $\Pi$ is estimated by frequency counting as the ratio of the number of times, $N(q\sigma)$, the state $q$ is followed (i.e., suffixed) by the symbol $\sigma$ and the number of times, $N(q)$, the state $q$ occurs; the details are available in [94]. The estimated morph matrix $\hat{\Pi}$ and the stationary state probability vector $\hat{P}(q)$ are obtained as:

$$\hat{\pi}(q, \sigma) \triangleq \frac{1 + N(q\sigma)}{|\Sigma| + N(q)} \quad \forall \sigma \in \Sigma \forall q \in Q; \quad \hat{P}(q) \triangleq \frac{1 + N(q)}{|Q| + \sum_{q' \in Q} N(q')} \forall q \in Q$$  \hspace{1cm} (4.2)

where $\sum_{\sigma \in \Sigma} \hat{\pi}(\sigma, q) = 1 \forall q \in Q$. Then, the D-Markov entropy rate (see Definition 4.4.2) is computed as:

$$H(\Sigma|Q) = -\sum_{q \in Q} \sum_{\sigma \in \Sigma} P(q)P(\sigma|q) \log P(\sigma|q) \approx -\sum_{q \in Q} \sum_{\sigma \in \Sigma} \hat{P}(q)\hat{\pi}(q, \sigma) \log \hat{\pi}(q, \sigma)$$

**State Merging:** While merging the states, this algorithm aims to mitigate this risk of degraded precision via a stopping rule that is constructed by specifying an acceptable threshold $\eta_{mrg}$ on the distance $\Psi(\cdot, \cdot)$ between the merged PFSA and the PFSA generated from the original time series. The distance metric $\Psi(\cdot, \cdot)$ between two PFSAs $K_1 = (\Sigma, Q_1, \delta_1, \pi_1)$ and $K_2 = (\Sigma, Q_2, \delta_2, \pi_2)$ is as follows:

$$\Psi(K_1, K_2) \triangleq \lim_{n \to \infty} \frac{1}{2^j + 1} \sum_{j=1}^{n} \|P_1(\Sigma^j) - P_2(\Sigma^j)\|_{\ell_1}$$  \hspace{1cm} (4.3)

where $P_1(\Sigma^j)$ and $P_2(\Sigma^j)$ are the steady state probability vectors of generating words of length $j$ from the PFSA $K_1$ and $K_2$, respectively, i.e., $P_1(\Sigma^j) \triangleq [P(w)]_{w \in \Sigma^j}$ for $K_1$ and $P_2(\Sigma^j) \triangleq [P(w)]_{w \in \Sigma^j}$ for $K_2$. States that behave similarly (i.e., have similar morph probabilities) have a higher priority for merging. The similarity of two states, $q, q' \in Q$, is measured in terms of the respective morph functions of future symbol generation as the distance between the two rows of the estimated morph matrix $\hat{\Pi}$ corresponding to the states $q$ and $q'$. The $\ell_1$-norm has been adopted to be the distance function as seen below.

$$\mathcal{M}(q, q') \triangleq \|\hat{\pi}(q, \cdot) - \hat{\pi}(q', \cdot)\|_{\ell_1} = \sum_{\sigma \in \Sigma} |\hat{\pi}(q, \sigma) - \hat{\pi}(q', \sigma)|$$
Hence, the two closest states (i.e., the pair of states $q, q' \in Q$ having the smallest value of $\mathcal{M}(q, q')$) are merged using the merging algorithm explained in [94]. The merging algorithm updates the morph matrix and transition function in such a way that does not permit any ambiguity of nondeterminism [110]. Subsequently, distance $\Psi(\cdot, \cdot)$ of the merged PFSA from the initial symbol string is evaluated. If $\Psi < \eta_{mrg}$ where $\eta_{mrg}$ is a specified merging threshold, then the machine structure is retained and the states next on the priority list are merged. On the other hand, if $\Psi \geq \eta_{mrg}$, then the process of merging the given pair of states is aborted and another pair of states with the next smallest value of $\mathcal{M}(q, q')$ is selected for merging. This procedure is terminated if no such pair of states exist, for which $\Psi < \eta_{mrg}$.

State transition matrix, denoted by $\Pi$ ($\Pi \triangleq [\pi_{ij}]$, $i = 1, 2, \cdots, |Q|$, $j = 1, 2, \cdots, |Q|$), is obtained via combining $\tilde{\pi}$ and $\delta$. Each element of $\Pi$, $\pi_{ij}$ is the probability of moving from state $q_i$ to $q_j$ upon occurrence of a symbol at the next time step. In this chapter, depth of the $D$-Markov machine is chosen to be one and it results in the equality of state transition matrix ($\Pi$) and probability morph matrix $\tilde{\pi}$. Depth greater than one can also be chosen via applying generalized $D$-Markov machine construction [123, 94]. $\Pi$ is considered as the output feature of the $D$-Markov machine, which represents the time-series in reduced dimension. More details on STSA can be found in [110, 123].

### 4.4.3 Cross modeling for sensor fusion

In $\times D$-Markov machines, the Markov assumption is made on the expected outcome of a symbolic process conditioned on the states of another symbolic process (instead of states of the same symbolic process). We assume that a symbol block of (finite) length $D$ is sufficient to describe the current state for the PFSA constructed from the symbol stream $\{s_1\}$. In other words, the symbols that occur prior to the last $D$ symbols do not affect the subsequent symbols observed in symbol stream $\{s_2\}$. We define a $\times D$-Markov machine next.
Definition 4.4.3 ($\times$D-Markov) Let $K_1$ and $K_2$ be the PFSAs corresponding to symbol streams $\{s_1\}$ and $\{s_2\}$, respectively. Then, an $\times$D-Markov machine (from $\{s_1\}$ to $\{s_2\}$) is defined as a 5-tuple $\mathcal{M}_{1\rightarrow 2} \triangleq (Q_1, \Sigma_1, \Sigma_2, \delta_1, \tilde{\Pi}_{12})$ such that:

1. $Q_1 = \{q_1, q_2, \ldots, q_{|Q_1|}\}$ is the state set corresponding to symbol sequence $\{s_1\}$

2. $\Sigma_1 = \{\sigma_1^0, \ldots, \sigma_1^{|\Sigma_1|-1}\}$ is the alphabet set of symbol sequence $\{s_1\}$

3. $\Sigma_2 = \{\sigma_2^0, \ldots, \sigma_2^{|\Sigma_2|-1}\}$ is the alphabet set of symbol sequence $\{s_2\}$

4. $\delta_1 : Q_1 \times \Sigma_1 \rightarrow Q_1$ is the state transition mapping for $M_1$

5. $\tilde{\Pi}_{12}$ is the cross morph matrix of size $|Q_1| \times |\Sigma_2|$; the $ij$th element $(\pi_{12}(q_i, \sigma_2^j))$ of $\tilde{\Pi}_{12}$ denotes the probability of finding the symbol $\sigma_2^j$ in the symbol string $\{s_2\}$ at next time step while making a transition from the state $q_i$ of the PFSA constructed from the symbol sequence $\{s_1\}$.

Analogous to entropy-rate of a PFSA, we define a cross-entropy rate for a crossed PFSA which we will use as a metric for constructing the states of the cross-PFSA model for two observed symbolic processes.

Definition 4.4.4 ($\times$D-Markov Entropy Rate [118]) The $\times$D-Markov entropy rate from a PFSA $(\Sigma_1, Q_1, \delta_1, \pi_1)$ to a symbol stream (say, $\{s_2\}$ with alphabet set $\Sigma_2$) is defined as:

$$H(\Sigma_2|Q_1) \triangleq \sum_{q_1 \in Q_1} P(q_1)H(\Sigma_2|q_1)$$

$$= - \sum_{q_1 \in Q_1} \sum_{\sigma_1^2 \in \Sigma_2} P(q_1)P(\sigma_1^2|q_1) \log P(\sigma_1^2|q_1) \tag{4.4}$$

where $P(q_1)$ is the probability of a PFSA state $q_1 \in Q_1$ and $P(\sigma_1^2|q_1)$ is the conditional probability of a symbol $\sigma_1^2 \in \Sigma_2$ given that a PFSA state $q_1 \in Q_1$ is observed.

The number of states of a $\times$D-Markov machine of depth $D$ is bounded above by $|\Sigma_1|^D$, where $|\Sigma_1|$ is the cardinality of the alphabet $\Sigma_1$. However, from the perspective of modeling the cross-dependance from $\{s_1\}$ to $\{s_2\}$, some states may be more important than others in terms of their embedded causal information contents. Thus it might be redundant to keep all states of
Figure 4.4: Variation of $xD$-Markov entropy rate directed from time-series of DBN hidden units (output of hi-speed video) to pressure time series as a function of number of state splitting at a symbol size $|\Sigma| = 3$ for (a) stable combustion, (b) thermo-acoustically unstable combustion.

different depths in the cross-PFSA model. Instead it might be advantageous to keep a set of states that correspond to symbol blocks of different lengths i.e., make a system where different states have different memory or depth $D$. This is accomplished by starting off with the simplest set of states (i.e., $Q_1 = \Sigma_1$ for $D = 1$) and subsequently splitting the current state that results in the largest decrease of the $xD$-Markov entropy rate $H(\Sigma_2|Q_1)$ (see Eq. (4.4)). Thus we can restrict the exponential growth of states with increasing depth $D$ without compromising the details of symbolic dynamics. At each step of state splitting, each element $\pi_{12}(\sigma_2, q_1)$ of the cross morph matrix $\Pi_{12}$ is estimated numerically by frequency counting as the ratio of the number of times, $N(q_1\sigma_2)$, the state $q_1$ from $\{s_1\}$ is followed by the symbol $\sigma_2$ from $\{s_2\}$ and the number of times, $N(q_1)$, the state $q_1$ occurs. A stopping rule for state splitting is constructed by specifying the threshold parameter $\eta_{spl}$ on the rate of decrease of $xD$-Markov entropy rate. The final estimated morph matrix $\hat{\Pi}_{12}$ can then used as representative feature of causality from the first to second symbolic process. This can then be used to perform various machine learning operations like pattern matching, classification, regression, clustering etc.

4.5 Early detection with DBN+STSA

The DBN used for the study is comprised of three hidden layers with 1000, 100, and 10 hidden units for the first, second, and third hidden layer respectively. The input image has a dimension of $56 \times 98$ pixels flattened to a $1 \times 5488$ row vector. The input image segments are
taken from respective images at the flame entry (right end of the images) zone after scaling the original images down by 4 times.

4.5.1 DBN feature visualization

For visualization, the training set consists of 54,000 training images containing 6,000 images each from 9 conditions, 9,000 validation images containing 1,000 images each from 9 conditions, and 18,000 test images containing 2,000 images each from 9 conditions. A learning rate of 0.01 is used for the gradient descent algorithm for both pre-training and supervised finetuning. Pre-training is performed in batches of 50 samples and each layer undergoes 30 complete iterations of pre-training before moving onto the next layer. During supervised finetuning, classification errors on the validation images is compared against the errors from training set as a measure to prevent overtraining the network and consequently overfitting the data. The optimized model is obtained prior to the point when the validation error becomes consistently higher than the training error in subsequent training iterations.

Figure 4.5: (d) Visualization of weights from the first layer and inputs that maximizes the hidden unit activations for the (c) 1st layer, (b) 2nd layer, and (a) 3rd layer after pre-training and prior to supervised finetuning.

Figure 4.5 (d) shows the visualization of weights from the first layer with each tile representing a hidden unit in the layer immediately after pre-training. Values of weights connecting from all visible units to this single hidden unit are represented as pixel intensities. Panels (c), (b), and
(a) visualize the input that maximizes the activation of the hidden units in the first, second, and third hidden layers respectively. As expected, the weights and the inputs that maximizes the activation of the first hidden layer are similar except that the pixel intensities are inverted. For higher layers, the network is able to capture the whole mushroom-shaped features from the input images. However, visualization for the third hidden layer (with only 10 hidden units) is not as clear due to the activation maximization algorithm converging to a non-ideal local optimum. A faint mushroom shape is still visible, however. In general, the pretrained model acquires a good representation of the input. Prominent features serving as the key to distinguishing between stable and unstable flames can clearly be seen in the visualized weight matrices.

![Visualization of weights from 1st layer and inputs](image)

Figure 4.6: (d) Visualization of weights from 1st layer and inputs that maximizes the hidden unit activations for the (c) 1st layer, (b) 2nd layer, and (a) 3rd layer after supervised finetuning.

In Figure 4.6, visualization of weights from the first layer and inputs that maximizes activations for all hidden layers after supervised finetuning are shown. An immediate difference can be clearly observed: visualized weights are now less noisy, whereas the third hidden layer is able to produce a visualization with more clarity compared to the weights prior to finetuning.

For both cases, the learning rate used in the AM algorithm is 0.01. Results have also indicated that depending on the initial value of the input vector, the resulting visualization from solving the optimization problem will be very different in terms of clarity. Thus, initial values of the input vectors are manually tuned by trial-and-error in order to obtain the best result. However,
random initialization of the input vectors over a uniform distribution yielded undesirable results most of the time, showing images that are completely noisy without any perceivable features. Even if the results do converge, there are no significant differences between the solution from random initialization compared to the solution from tuning the initial values manually.

**Remark:** It is observed from the feature visualization that, though the DBN is trained on both stable and unstable flame images, the features gravitate more towards the coherent structure which is a characteristic of thermo-acoustic instability. An expert can use this feature visualization as an important tool to choose templates for unstable combustion, especially from the higher layer features. Those templates can be applied in post-processing of images to calculate the extent of instability via appropriate metrics that can effectively replace the age-old need for hand-crafted visual features.

### 4.5.2 Performance of STSA module

In this subsection, DBN is pre-trained with 36,000 training images coming from 4 different operating conditions (see table 4.1) at partial premixing. Half of the training data is collected during stable combustion and other half during unstable combustion. Two sequences of images, consisting of one at stable ($Re = 7,971$, $FFR = 0.495g/s$ and full premixing) and another at
unstable ($Re = 15,942$, $FFR = 0.495g/s$ and full premixing) combustion states, are reduced
dimensionally via DBN with the parameters learned at pre-training phase. It is to be noted
that, pre-training and testing of DBN are done on data at different levels of premixing to test
the transferability of the proposed architecture.

Time series of $l_2$ norm of 10 dimensional activation probability vectors from each image
are obtained as shown in Fig. 4.7(c) and (d). For comparison, $l_2$ norm of 10 largest variance
components of those images, based on principal component analysis (PCA) [18] coefficients
learned on same training images, are constructed as presented in the top half of the Fig. 4.7.
It is observed that the difference in textures of the $l_2$ time series between stable and unstable
combustion is amplified in the case of DBN feature learning.

STSA is performed with increasing alphabet size on the $l_2$ time series that are mentioned
above. Time series for stable and unstable combustion are partitioned separately via MEP
and respective state transition matrices are calculated by the method explained in section 4.4.
Euclidean distance between state transition matrices of stable and unstable combustion is a
measure of class separability between those. The more the class separability is the more would
be the precision of detecting the intermediate states of the combustion while shifting from stable to unstable state. Therefore, this framework is better suited for early detection of onset of instability. It is presented in Fig. 4.8 that the class separability is much higher when STSA is applied on pre-trained DBN features than the PCA features. A probable rationale behind this observation is that, while PCA is averaging the image vector based on just maximum spatial variance, DBN is learning semantic features based on the coherent structures seen during unstable combustion. This rationale is also supported by the DBN feature visualizations that are shown in the subsection 4.5.1.

In a PHM context, the state transition matrix emerging from STSA module at the top can be used in supervised manner to detect instability from hi-speed image data. As the ‘DBN+STSA’ architecture provides a large class-separability between stable and unstable conditions, the state transition matrix can help in early prediction of thermo-acoustic instability. While the training of the proposed architecture is carried out offline in a GPU, the testing in a PHM application can be performed online with a processing power of a regular CPU. This is possible because the feed-forward computation of DBN along with STSA is feasible in real-time.

4.6 Early detection with CNN+STSA

4.6.1 CNN training

The network is trained using flame images with 4 different unstable combustion conditions mentioned in the section 4.2. The data consists of 24,000 examples for training and 12,000 examples for cross-validation. In the first convolutional layer, 20 filters of size $33 \times 34$ pixels (px) reduce the input image of dimension $100 \times 237$ pixels to feature maps of $68 \times 204$. Next, the feature maps are downsampled with a $2 \times 2$ max-pooling, resulting in pooled maps of $34 \times 102$ px. Each of these maps undergoes another convolutional layer with 50 filters of $23 \times 23$ px which produces feature maps of $12 \times 80$ px (before $2 \times 2$ max-pooling), and $6 \times 40$ pooled maps after max-pooling. All generated maps are connected to the fully connected layer of 100 hidden units followed by 10 output units where the sigmoid activations are extracted. Training is performed with a batch size of 20 and learning rate of 0.1. Convolution is done without any padding with
a stride-size of 1. Visualization of few filters at first and second convolutional layer is shown in Fig. 4.9 (a), (b). Second layer visualization shows that it captures fragments of the flame coherent structures. Fig. 4.9 (c) presents couple of the feature maps of a stable frame (top) and an unstable frame (bottom) after convolving with first layer filter. Red outline at the bottom exhibits how the mushroom-shaped coherent structure is highlighted on the unstable frame feature map.

Figure 4.9: Filter visualization at convolutional layer (a) one and (b) two. (b) shows fragmented representations of coherent structures that are visible in unstable flame. (c) Feature maps of a stable frame (top) and an unstable frame (bottom) after applying first convolutional layer filter. Red outline on the unstable flame visualization shows how the mushroom-shaped coherent structure is highlighted

4.6.2 STSA-based instability measure with CNN

Once the CNN is trained on the sets of unstable data, every frame of the transition data sets (i.e., \(500_{40\to38}, 500_{40\to30}, 40_{500\to600}, 600_{50\to35}\) and \(50_{700\to800}\) as mentioned in section 4.2) are fed to the CNN. Each sigmoid activation unit out of ten at the last fully connected layer generates a time series for one transition data set. For capturing the fast change in a transition data, a window of 0.5 seconds (1500 frames) is traversed over the hi-speed video with an overlap of 80% to keep the response speed at 10 Hz, which is necessary for real-time combustion instability control. The time window output of a sigmoid activation unit is symbolized by maximum entropy partitioning (MEP) with an alphabet size of \(|\Sigma| = 3\). Considering the first window to be reference stable state, a generalized D-Markov machine is constructed by state splitting with \(N_{max} = 10\) and state merging with \(\eta_{mrg} = 0.05\). \(N_{max}\) is chosen as 10 because window length is not enough to learn a large state machine. For the alphabet \(\{1, 2, 3\}\), the set of states after state splitting is \(\{11, 21, 31, 2, 113, 213, 313, 23, 133, 233, 333\}\) and state merging leads to
\{11, 21, 31, 2, \{113, 313\}, 213, \{23, 133\}, \{233, 333\}\} for one of the sigmoid activation outputs in the transition video 600_{50 to 35}. State probability vector, arising from D-Markov machine at each time window, is the feature capturing the extent of instability which is transmitted through the corresponding sigmoid hidden unit. Instability measure of a time window is defined as the \(l_2\) norm distance from the reference stable time window.

![Variation of the proposed instability measure with time for the transition video named 600_{50 to 35}.](image)

Figure 4.10: Variation of the proposed instability measure with time for the transition video named 600_{50 to 35}. Multiple regions on the measure curve denote different combustion states such as stable, temporary intermittancy (a significant precursor to persistent instability) and unstable. They are corresponded to varied coherent structures (bounded by red box) that are detected by the ‘CNN+STSA’ framework. On the right, \(\text{rms}\) variation of the pressure is shown as it is one of the most commonly used instability measures. Progression of \(P_{\text{rms}}\) can not detect the aforementioned precursors.

Fig. 4.10 shows an aggregated progression (summation of individual instability measure obtained from each sigmoid activation unit) of instability measure for 600_{50 to 35}. The \(\text{rms}\) curve of the pressure on right of the Fig. 4.10 gives a rough idea about the ground truth regarding stability. Two fold advantages of the proposed instability measure over \(P_{\text{rms}}\) are as follows: (i) intermittancy phenomenon (region 2 and 3 on Fig. 4.10) is captured by this measure because it can detect variable-size mildly-illuminated mushroom-shaped coherent structure (bounded by red box in the Fig. 4.10) in the ‘CNN+STSA’ framework whereas \(P_{\text{rms}}\) ignores these important precursors to instability and (ii) region 4 of the Fig. 4.10 shows that the proposed measure rises faster towards instability. Other transition data sets also exhibit similar nature regarding this measure. Hence, the proposed measure performs better in early detection of instability than other commonly used measures such as \(P_{\text{rms}}\).
4.6.3 Comparison of CNN+STSA with PCA+STSA

To compare with the proposed approach, Principal Component Analysis (PCA) [18], a well-known dimensionality reduction technique is used as a replacement of CNN module. Fig. 4.11 (a) shows that the transition (stable to unstable) increment of aggregated instability measure for ‘CNN+STSA’ is larger than that for ‘PCA+STSA’ in all transition data. This will result in more precise instability control in real time. The condition $50_{7000 to 800}$ is observed in Fig. 4.11 (b) as the transition jump for both frameworks are very close. A close observation of the instability measure variation reveals that ‘CNN+STSA’ can detect an intermittancy precursor (region 1 at Fig. 4.11 (b)) although the coherent structure formation is not very prominent. However, ‘PCA+STSA’ misses this precursor before arriving at the inception of persistent instability. A probable rationale behind this observation is that, while PCA is averaging the image vector based on just maximum spatial variance, CNN is learning semantic features based on the coherent structures of varied illumination, size and orientation seen during unstable combustion.

Figure 4.11: (a) Comparison of sudden change in instability measure when instability sets in for different transition conditions which are 1. $500_{40 to 38}$, 2. $500_{40 to 30}$, 3. $40_{5000 to 600}$, 4. $600_{50 to 35}$ and 5. $50_{7000 to 800}$. The jump is larger for ‘CNN+STSA’ than ‘PCA+STSA’. (b) Variation of instability measure for both ‘CNN+STSA’ than ‘PCA+STSA’ at transition condition $50_{7000 to 800}$. The measure arising from ‘CNN+STSA’ can detect the intermittancy precursor whereas it is mostly ignored by ‘PCA+STSA’. A frame with an intermittancy coherent structure in a red box is shown on the top.
4.7 Early detection with multimodal sensor fusion

STSA learns D-Markov machines via symbolization and probabilistic finite state automata (PFSA) [110] generation from pressure data and individual time series arising from DBN bottleneck hidden units. It also models variable depth $\times D$-Markov machines which captures the cross-dependence among DBN hidden layer temporal output and pressure time series. This paper proposes to exploit this $\times D$-Markov machine construction for early detection of thermo-acoustic instability via spatiotemporal fusion. While the DBN efficiently reduces the dimension of the hi-speed video by preserving the semantic feature (continuing presence of coherent structures [120] in hidden layer visualization), STSA models temporal evolution of that feature via constructing $\times D$-Markov machine that considers heterogeneous sensors spatially dislocated. Elements of the proposed tool chain are explained in detail later in the sequel.

This section discusses the results that are exhibited when the proposed framework is applied on the experimental data for early detection of thermo-acoustic instability via spatiotemporal fusion of hi-speed video and pressure data. For learning the DBN from video data, a network with three hidden layers of size 1000, 100 and 10 respectively is chosen while keeping the volume of training data under consideration. After scaling down by 4, the $56 \times 98$ pixels frame at the entry of the flame serves as the input vector ($1 \times 5488$ after flattening) which is fed into the DBN at each time instant. Data from all four conditions at partial premixing ($X_1 = 90mm$) and two conditions at full premixing ($X_1 = 90mm$, $FFR = 0.495g/s$, $Re = 7,971; 15,942$) containing 54000 images in total are used for training the DBN via stochastic gradient descent method as explained in section 2.1.2. For both pre-training and supervised finetuning, learning rate of 0.01 is used for the gradient descent algorithm. This study mainly emphasizes on the last hidden layer output after pretraining because previous work by the authors [120] suggested that even pretrained hidden layers capture significant amount of coherent structures in the flame without supervised learning. Eventually, successful modeling of flame coherent structure is important during dimensionality reduction of images for constructing a scaler measure that detects instability early.
Once the DBN is learned via layer-wise pretraining, test video data containing three operating conditions at full premixing are fed into the network to obtain the last hidden layer representation by feed-forward operation. The details of the three conditions are as follows: (i) $FFR = 0.66g/s$, $Re = 10,628$ at stable state, (ii) $FFR = 0.083g/s$, $Re = 1,771$ at relatively stable state and (iii) $FFR = 0.308g/s$, $Re = 10,628$ at thermo-acoustically unstable state. During the testing phase, temporal sequence of high dimensional images (i.e., 5488 pixels) generates multi-variate time series with few hidden unit ($= 10$) activation probabilities at bottleneck hidden layer of the DBN. Thus, fusion of large dimensional hi-speed video and pressure time series is reduced to a problem of fusing the DBN bottleneck layer time series and pressure time series.

The STSA module constructs generalized D-Markov machine [124] from individual time series and models the cross-dependence from one time series to another by constructing $\times D$-Markov machine. $\times D$-Markov machines from bottleneck layer time series (for each hidden unit) to synchronized pressure time series are obtained by the process explained in section 4.4.3. The direction of $\times D$-Markov machine construction is identified to be from hi-speed video bottleneck layer to pressure data because pressure sensor is located downstream of the imaging sensor according the flow. The time series for both modalities are symbolized by maximum entropy partitioning (MEP) [110] with an alphabet size of $|\Sigma| = 3$. Fig. 4.4 shows the drop of $\times D$-Markov entropy rates with increasing number of state splitting while constructing the $\times D$-Markov machines directed from bottleneck layer hidden units to pressure time series. For stable state, $\times D$-Markov entropy rate doesn’t decrease significantly to a definite knee point as the time series itself is highly chaotic [99]. Based on $\eta_{spl}$, a cross-PFSA model can be selected in this scenario. Right half of Fig. 4.4 shows that $\times D$-Markov entropy rate for 10th hidden unit to pressure time series converges earliest among others after 12 splits (i.e., 27 state $\times D$-Markov machine). Based on this analysis, the $\times D$-Markov machine directed from 10th hidden unit to pressure data can be chosen for further fusion operations.

When stable combustion becomes thermo-acoustically unstable, chaotic pressure time series becomes hi-amplitude periodic tones and phase differences among spatially apart sensors reduces drastically [108]. To capture this variation of temporal complexity and spatial dynamics among combustor sensors during the onset of instability, $\times D$-Markov entropy rate is proven to be a
Figure 4.12: Entropy rate as an instability measure at stable, relatively stable and unstable states for (a) D-Markov machine on pressure time series, (b) $\times D$-Markov machine directed from hi-speed video dominant-PCA-feature-aggregate time series to pressure time series, (c) $\times D$-Markov machine directed from hi-speed video bottleneck layer 5th hidden unit to pressure time series and (d) 10th hidden unit to pressure time series.
good candidate for constructing a sensitive and robust measure [116]. As instability creeps in, \( \times D \)-Markov entropy rate reduces because temporal complexity and phase difference deteriorate. Entropy rates are calculated on every 0.3 seconds window with 0.2 overlap based on the model learned via state splitting. There are 3 sec long temporal data coming from each bottleneck hidden unit and pressure time series for either of stable, relatively stable and unstable states. Computational complexity of the proposed approach is adequate to meet the requirement for combustion instability control, which is 10Hz response speed. Fig. 4.12(a) presents a boxplot exhibiting a premature drop of pressure entropy rate along with a high variance over multiple time windows. Thus, entropy rate based on only pressure data may cause severe false alarm while detecting early onset of instability. This observation motivates the requirement of multi-modal fusion for this purpose. To compare with the proposed approach, Principal Component Analysis (PCA), a well-known dimensionality reduction technique is used as a replacement of DBN module. Aggregate time series of 10 largest variance components of test images are extracted based on PCA coefficients learned on same training images. It is observed in the boxplot of Fig. 4.12(b) that \( \times D \)-Markov entropy rate directed from video generated PCA-aggregate time series to pressure time series is performing better than pressure entropy rate. However, it still has a high variance and low median at relatively stable state, which might cause false alarm in online prediction. Figures 4.12(c), (d) show the boxplots of \( \times D \)-Markov entropy rate directed to pressure time series from hi-speed video bottleneck layer 5th hidden unit and 10th hidden unit respectively. It is observed that the variance over time window is lower (especially for 10th hidden unit) compared to top half of the figure, making this measure more robust for real-time instability control. Overall, the negative slope of entropy rate drop for 'DBN+STSA' is higher than that of pressure or 'PCA+STSA' approaches. These observations denote that the \( \times D \)-Markov rate obtained via proposed 'DBN+STSA' fusion approach is a sensitive and robust measure for early detection instability.

4.8 Early detection using deep convolutional selective autoencoders

Here, we study the use of an end-to-end deep convolutional selective autoencoder approach to capture the rich information in hi-speed flame video for instability prognostics. In this context,
an autoencoder is trained to selectively mask stable flame and allow unstable flame image frames. The network identifies subtle instability features as a combustion process makes transition from stable to unstable region. The proposed framework is validated on a set of real data collected from a laboratory scale combustor over varied operating conditions. As a result, the deep learning tool-chain can perform as an early detection framework for combustion instabilities that will have a transformative impact on the safety and performance of modern engines. Whereas traditional PHM algorithms use mainly time series data (e.g., pressure and temperature etc.) for this purpose, the proposed approach attempts to advance PHM via capturing the rich information of hi-frequency video. The approach performs implicit labeling in order to derive soft labels from extreme classes that are explicitly labeled as either positive or negative examples. This particular property is significant for tracking continuous temporal phenomenon such as the transition from combustion stability to instability, where labels of extreme states (stable or unstable) are available but intermediate state labels are not. Explicit labels are utilized to selectively mask selective features while allowing other features to remain. Fig. 4.13 shows grayscale images describing typical gradual development of instability at the stated parameters in the swirl-stabilized combustor used for the experiment.

![Grayscale images of gradual time-varying development of instability structure at two different parameter values](image)

Figure 4.13: Grayscale images of gradual time-varying development of instability structure at two different parameter values

Labeling (e.g., structured and implicit) can be considered a multi-class classification problem [35]. For example, three-stage Hidden Markov Models (HMM) were used for handling speech recognition [106] problems, parts of speech tagging [89] and sequence labeling because they derive
the relationships from observations to state and state to state in dynamic systems. Maximum Entropy Markov Model (MEMM), a discriminative modification of HMM, was introduced to overcome the latter’s recall and precision problems especially in labeling texts. In those models, conditional probability of the desired labels are learnt directly based on the uncertainty maximization idea. Applications of MEMM for natural language processing can be found in [14].

Due to “label bias” defects of MEMM, a Conditional Random Field (CRF), which is a joint Markov Random Field (MRF) of the states conditioned on the whole observations was later explored by [69]. It enabled considering the global labels of the observation as against localization of labels of MEMM [35]. However, labeling in this case is made computationally complex by the relaxation of statistical independence assumption of the observations which most of the models assume.

Recurrent Neural Networks (RNNs) have been utilized for sequence labeling problems due to its cyclic connections of neurons [45] as well as its temporal modeling ability. Although earlier construction of RNNs was known to have short ranged memory issues and a restrictive unidirectional information context access, formulation of a bidirectional Long Short Term Memory (LSTM) [46] resolved such issues. However, this construction adds to the complexity of the model significantly as typically two RNNs get connected through the same output layer.

From the application standpoint, early detection of instability in the combustion chambers of dynamic systems aids anticipative actions for reducing its consequent effects. Visualizing the features that characterizes the intermediate frames of its spectrum is an important approach to unravel the processes that precede instability. The authors in [120] introduced Deep Belief Networks (DBN) as a viable technique to achieve the aim with a view to exploring other machine learning for confirmation.

The instability detection problem actually boils down to an implicit soft labeling problem where we train a deep model using hi-speed flame videos with explicit labels of stable and unstable flames such that it recognizes the onset of instability early as the combustion process makes transition from a stable to unstable region. Conceptually, this is similar to cognitive psychologists’ description of human reasoning in object classification [140]. An example is to consider how a child is taught on intrinsic classes. A similar problem is how to detect a cross
breed of dog and wolf and how close the animal is to either of the classes. From an application standpoint, an early detection of engine’s combustion instability may be useful for computing the instantaneous values of the remaining useful life, but the computation is partial since other engine physical use factors are also important. Therefore, RUL computation is beyond the scope of the present problem.

4.8.1 Convolutional Selective Autoencoder

Based on the performance of CNN on several similar tasks reviewed, it was found a suitable candidate for the composite architecture to examine our hypothesis of soft label generation. A CNN architecture for low-level feature extraction with a symbolic graphical model such as STSA at the top level [121] has been used in the previous section. In contrast, we use an end-to-end convolutional selective auto-encoder (as shown in Fig. 4.14), designed and tested in [5], to explore another perspective to the current problem.

![Figure 4.14: Structure of the convolutional autoencoder with associated layer parameters. The encoder portion extracts meaningful features from convolution and sub-sampling operations, while the decoder portion reconstructs the output into the original dimensions through deconvolution and upsampling. Unpooling is done with \(2 \times 2\) upsampling by replicating the elements. Best viewed on screen, in color.](image)

Given an \(M \times N\) dimensional image frames and corresponding ground truth labels (one of the two classes), explicit labels are generated by selectively masking frames with the undesired
class with black pixels. Hence, \( N \) pairs of input-output pairs \( \{(X_i, Y_i)\} \) for \( i = 1, 2, ..., N \) are generated where \( X \) represents the original images, \( Y \) are the masked frames that are considered explicitly as ground truth. The images are then normalized where pixel intensities have zero mean and a standard deviation of 1 as preprocessing.

### 4.8.2 Instability measure

The similarity index selected for instability measure is the correlation ratio reported in [43] and mathematically proven by [76] to have low computational requirement. Moreover, its ability to quantify the relationship between two image frames with differing intensities as well as not directly including the actual images in the computation is quite useful. Assuming that the input image \( X \) has pixels \( x \in X \) with intensities \( x_i \in [0, 255] \) for \( i = 1, 2, ... \) and the inferred output frames denoted as \( Y \), the correlation ratio is calculated by computing the conditional variance, 

\[
\sigma_i = \frac{1}{Z_i} \sum_{x_i} (Y[x_i])^2 - \left( \frac{1}{Z_i} \sum Y[x_i] \right)^2
\]

and the total variance \( \sigma = \frac{1}{Z} \sum X(Y[x])^2 - \left( \frac{1}{Z} \sum Y[x] \right)^2 \) in \( Y \). The instability measure thus becomes \( \delta = \frac{1}{Z} \sum_i Z_i \sigma_i \) where \( Z_i \) and \( Z \) are enumerations of the pixels in \( x_i \) and \( X \) respectively. Like other chosen measure of dissimilarity such as \( \ell_1 \) and \( \ell_2 \) norm, the correlation ratio usually varies from 0 for uncorrelated images and 1 for fully correlated images.

### 4.8.3 Implementation

**Training process:** The architecture inputs and implementation for model learning from data by the architecture are described in this part. In training the network, 63,000 grayscale frames having dimensions 100 \( \times \) 237 are resized to 64 \( \times \) 64 for computational simplicity. A total of 35,000 frames was labeled stable while the remaining 28,000 were labeled unstable. These images were a combination of datasets with different premixing lengths of either 90mm or 120mm and a wide range of air and fuel LPMs for which the combustor is either in a stable or an unstable state. A learning rate of 0.0001 with momentum = 0.975 was found to train the model best in the Nesterov based stochastic gradient descent formulation. The network was trained to 100 epochs in order to conveniently strike a good minima of the validation error. As
stated earlier, $\ell^2$—regularization parameters of 0.0001 each were added to widen the parameter search space for locating the minima by helping to minimize the difference between the test and training. Also, the $\ell^1$ enhances the sparsity of the algorithm while ensuring that only the most likely units are activated. Training was done on GPU Titan Black with 2880 CUDA cores, equipped with 6GB video memory, using the python-based machine learning frameworks such as Theano, Lasagne and NoLearn [16, 141]. Lasagne offers a wide variety of control over the layer types, nonlinearity types, objective functions, interfacing with Theano, and many other features built into it. NoLearn, on the other hand, is a coordinating library for the implementation of the layers in Lasagne which offers model visualization features. While training, a filter of $c \times c$ pixels ($c = 3$ in the implementation) and a non-overlapping $p \times p$ ($p = 2$) maxpooling were found to be experimentally less costly to produce the results. Algorithm training was done in batches of 128 training examples which was found to be suitable via cross validation. The architecture in

![Figure 4.15: Schematics of implementation of trained network on transition test data](image)

Fig. 4.14 shows how the layers are interlinked in the training stage which leads to an overall of 5,090,249 learnable parameters. Leveraging the capability of GPU, generating results from the trained model based on a transition sequence of 21,841 frames took $\approx 35.5\text{secs}$. Note that the computation time can be greatly reduced with dedicated Field Programmable Gate Arrays (FPGA) during actual implementation.
4.8.4 Performance Evaluation

In this section, some results obtained from the algorithm are discussed and analyzed closely. First, we consider the detection of the presence of region two properties in frames supposedly of region 1 in the early instability detection paradigm. Then we discuss how the network explores the space between the stable and unstable regions to get softer labels assuming the system were static.

**Early detection of combustion instability**: Let the stable region be denoted by SR on one end of the spectrum and the unstable region be UR on the other end of the spectrum. For emphasis, training of the algorithm was done with explicitly available ground truth labels. These were categorized into frames of stable flame types and frames of unstable flame types. Any unit of frames in the stable region are then given labels of ‘0’ while those of the unstable region were retained in algorithm training. Fig. 4.16 shows the algorithm’s ability to reproduce such training in one of the frames trained on. The algorithm’s selective ability is shown by Fig. 4.16.

![Figure 4.16: Illustration of the pipeline’s ability to reproduce explicit labels.](image)

Feature maps from the model are shown in Fig. 4.17 to highlight the detected features and the reconstructed outputs. Some important feature maps are visualized in in Fig. 4.17. The fully
connected layers serve at least two important purposes, namely: (1) to reduce further the image dimensions towards only rich explanatory features, and (2) ensuring structural consistency for optimal layerwise features by reshaping the output images into dimensions similar to the input.

For frames in the unstable region, the corresponding feature maps showed more activated units responding to the mushroom structures characteristic to unstable combustion. These are highlighted in Fig. 4.17. For those from the stable region, information is seen to be rapidly diffusing from the input into the hidden layers. At each layer, joint parameters capture the trade-off between discarded and retained information from the stable and unstable training sets. Based on the understanding so far, it is hypothesized that the trained model could identify frames having flame types intersecting both stable and unstable regions as discussed in the motivating example. The subsequent parts of the section are devoted to this analysis.

The results of evaluating the trained model on test data that were collected under different transition conditions are presented in Fig. 4.18. The figure shows the capability of the model to suppress features of frames in the stable region and reveal the desired anomaly features in the stable flame images. The similarity measure introduced in section 4.8.2 was used to evaluate the strength of the algorithm’s ability to mask examples closer to the stable region compared to those nearer to the unstable region. Thereafter, a local regression smoothing was applied to obtain weighted moving averages for visualizing the transitional trends.

The general trends and fluctuations in instability measures are shown in Fig. 4.18. These
Figure 4.18: Results of transition protocols for: a) $600_{50\rightarrow35}$, b) $500_{40\rightarrow30}$ and c) $50_{700\rightarrow800}$ where purple arrows indicate expected results from stability and instability while the green arrows with different intensities indicate the strength of early instability presence in a supposedly stable region.

Results are similar to those reported in [121] where the framework used a neural-symbolic approach with a combination of convolutional neural networks and symbolic time series analysis to obtain instability measures. For emphasis, no background knowledge is provided other than domain experts knowledge of short-time instability bursts in the stable region prior to instability. In Fig. 4.18(a), stable region’s output frame is suppressed while that in the unstable region is entirely visible. Essentially, the model has become a filter which only enables desired features to show up in the outputs. In these regions, the train model produces an output that partially reveals the unstable features as highlighted in Fig. 4.18(b). The same phenomenon can be observed in Fig. 4.18(c). A worthy mention is that intermittency is a precursor to combustion instability. As reported in [121], these intermittency phenomena can not be observed from pressure data or POD analysis of image data. Hence the proposed metric localizes the intermittencies prior to full-blown instability with more prominence than other state-of-the-art approaches [121]. Improved accuracy in tracking intermittence leads to an early detection of instability with less false alarm [121].
Frame labeling: A computationally complex decision in attempting implicit labeling with these results would be to search through all the frames for adjacent neighbors to a given frame. In clear terms, this means finding which frames come before and after any chosen frame. This kind of search is usually difficult with most primitive low dimensional local labeling algorithms (e.g. HMM and MEMM) due to dependency depth and labeling bias limitations respectively. We hypothesize that simplifying any such high dimensional problem using a single value, average instability measure through the composite convolutional auto-encoder would facilitate soft labeling. This is especially required in the regions before and after the abrupt transitioning from SR to UR. With averagely linear lines superimposed on the frames that are nearer to SR and UR, the complex topology of the frames’ arrangement could be simplified to a linear plot that enhances the determination of most likely nearest neighbor frames to any given reference frame.

Figure 4.19: Adjacency labeling result of transition protocols for: a) 600 to 35, b) 500 to 30 and c) 50700 to 800 with three consecutive frames on a transition protocol highlighted.
A proposed approach for adjacency labeling in transition protocols of Fig. 4.18 are shown in Fig. 4.19. By considering the averaged sections before and after the abrupt transition, the gradual labels could be explored for implicit neighborhood graph decision. Note that the data in this case had to be smoothened to remove the transients introduced by the dynamics of the combustor. However, the richness in the dynamics signifies some defect in this approach for labeling problems of generic applications.

Let the red lines be the average linear lines on each examples in Fig. 4.19 for a static system with similar result to that considered. The results of transition protocol in Fig. 4.19(a) is closest to the explicit labels example used for training the algorithm because the lines are constant before sudden jump. Also, in this case, changes between neighboring frames are more abrupt compared to those in Fig. 4.19(b) and Fig. 4.19(c) whose experimental protocols (e.g., changes in air-fuel ratio) represent more steady rise in the transition property. However, the experimental protocol for the case presented in Fig. 4.19(c) has more subtlety between the SR and UR with a fuzzy/noisy transition region. The closest representation of the implicit labeling problem among the datasets is that shown in Fig. 4.19(b). Given any frame at random, one would be able to determine all its sets of neighbors in both regions by a gradual change in the average instability measure especially after a random shuffling of the frame positions or in cases where the knowledge of ground truth states for each individual frames are not available. Also implicit labeling helps in higher level learning of the flame dynamics from the video by using probabilistic graphical model such as HMM or STSA [121].

Specifically highlighted on panels of Fig. 4.19(b) are the results of three consecutive frames on a region that would represent our hypothesis in the portions before and after full transition for static applications. In this application however, the gradual build up of the two lobes “mushroom-like structure” [120] from frame to frame as well as increasing average instability measure are a pointer to the gradual transitioning ability of the technique. Note that the dynamics of the regions in red line would hide this information if frames were selected from there. Thus, we could easily fix frames back to their original position in static applications by the soft labels generated. However, it is expected that the discriminative advantage of the network would intuitively support that presumption. This is a consequence of marginalizing on
each hidden layer given the previous layer which has aided determining roughly, neighboring flame patterns and thus providing a coarse to finer labels.

Computation time: In the present application, the average inference stage computation time for the 5 available transition data conditions was found to be 102s for 21841 frames. This is equivalent to about 4.7ms/frame and it was obtained on our 1458.843MHz, multiCPUs Linux 64bits computer. The convolutional selective autoencoder is in the class of algorithms with larger learning capacity leading to increased computation time. Again, the computation time can further be reduced with the use of dedicated FPGAs where the computation is parallelized.

4.9 Conclusion and Future Work

The paper proposes a framework that synergistically combines the recently introduced concepts of DBN, CNN, and STSA for early detection of thermo-acoustic instability in gas turbine engines. Extensive set of experiments have been conducted on a swirl-stabilized combustor for validation of the proposed method. Sequences of hi-speed greyscale images are fed into a multi-layered deep network to model the fluctuating coherent structures in the flame, which are dominant during unstable combustion. Hidden layers of the DBN, along with bottom layer weight matrix are visualized via activation maximization method and mushroom-shaped vortex are demonstrated by higher layers after, both, pre-training and finetuning stages. Although visualization after fine tuning is less noisy, it may lead to overfitting due to limitation of the data volume. Therefore, an ensemble of time series data is constructed from sequence of images based on the $l_2$ norm of the activation probability vectors of last hidden layer at the DBN. Then, STSA is applied on the time series that is generated from an image sequence and ‘DBN+STSA’ is found to exhibit more class separability with varying alphabet size than ‘PCA+STSA’. More class separability between stable and unstable combustion implies more precision at detecting early onset of thermo-acoustic instability. On the other hand, ‘CNN+STSA’ is found to exhibit larger change in instability measure while transition to instability than ‘PCA+STSA’. The proposed framework detects all the intermittent precursors for different transition protocols, which is the most significant step towards detecting the onset of instability early enough for mitigation. In summary, while the deep network captures the semantic features (i.e., coherent
structures) of the combustion flames, STSA models the temporal fluctuation of those features at a reduced dimension.

One of the primary advantages of the proposed semantic dimensionality reduction (as opposed to abstract dimensionality reduction, e.g., using PCA) would be seamless involvement of domain experts into the data analytics framework for expert-guided data exploration activities. Developing novel use-cases in this neural-symbol context will be a key future work. Some other near-term research tasks are:

1. Dynamically tracking multiple coherent structures in the flame to characterize the extent of instability,
2. Multi-dimensional partitioning for direct usage of the last sigmoid layer, and
3. Learning CNN and STSA together.

Additionally, an end-to-end convolutional selective autoencoder is developed to generate fuzzy labels from prior knowledge of hard labeled examples. The framework is used to perform early detection of combustion instabilities using hi-speed flame video. Validation results are presented using data from a laboratory scale swirl-stabilized combustor. The results are discussed in the light of a high fidelity similarity metric which is used to gauge the closeness to ground truth unstable flames. Using the same measure, the architecture was extended to address the neighborhood implicit graph labeling problem. The framework can be generalized for a high-dimensional data in order to perform soft-labeling by interpolation of explicit labels. While the framework is shown to be an efficient diagnostics technique for combustion process in laboratory experiments, large scale validation is underway to demonstrate its wide-range applicability. Some of the future works are:

1. To compare the performance of the proposed approach to competing labeling techniques such as CRF,
2. To extend framework to multiclass labeling scenarios, and
3. To compute the instantaneous estimate of the RUL by holding other factors constant.
PART II

DEEP LEARNING FOR AUTONOMOUS COMPLEX SYSTEMS
CHAPTER 5. DEEP VALUE OF INFORMATION ESTIMATORS FOR COLLABORATIVE HUMAN-MACHINE INFORMATION GATHERING

Effective human-machine collaboration can significantly improve many learning and planning strategies for information gathering via fusion of ‘hard’ and ‘soft’ data originating from machine and human sensors, respectively. However, gathering the most informative data from human sensors without task overloading remains a critical technical challenge. In this context, Value of Information (VOI) is a crucial decision-theoretic metric for scheduling interaction with human sensors. We present a new Deep Learning based VOI estimation framework that can be used to schedule collaborative human-machine sensing with computationally efficient online inference and minimal policy hand-tuning. Supervised learning is used to train deep convolutional neural networks (CNNs) to extract hierarchical features from ‘images’ of belief spaces obtained via data fusion. These features can be associated with soft data query choices to reliably compute VOI for human interaction. The CNN framework is described in detail, and a performance comparison to a feature-based POMDP scheduling policy is provided. The practical feasibility of our method is also demonstrated on a mobile robotic search problem with language-based semantic human sensor inputs.

5.1 Motivation

The notion of human-machine interaction for cooperative problem solving has attracted much interest in various cyber-physical system domains. Much of the human-machine interaction literature in the AI, robotics and controls communities tend to focus on the idea of humans acting in the role of collaborative planners or controllers for machine counterparts. In this work, however, we examine the problem of using humans as ‘soft data sensors’ for intelligent
machine systems. In particular, we focus on the question of how human observations can be used to augment state estimates of measurable dynamical physical states that must be monitored continuously by conventional ‘hard’ sensor data used by machines (object position, velocity, attitude, temperature, size, mass, etc.).

Several modeling approaches have been developed to exploit soft human sensing across a variety of interfaces, e.g. verbally reported range and bearing for target localization [58]; verbal and sketch-based detection/no detection reports for target search [20, 3]; and semantic language inputs for target localization [4]. While these works have largely focused on developing human sensor models and suitable data fusion algorithms for blending hard and soft data, relatively little work has been done on active soft sensing, i.e., intelligent querying of human sensors to gather information that would be most beneficial for complex machine planning and/or perception tasks. Active sensing problems have a rich tradition in target tracking and controls communities, but have focused on hard data sources such as radar, lidar, cameras, etc. One particularly relevant issue is the sensor scheduling problem, i.e., the selection of most appropriate sensing assets given constraints on how many can be tasked to deliver data any given instant (e.g., due to bandwidth or computational limits). In this work, we address the problem of scheduling interactions between human sensors and their machine counterparts: what information should be solicited from human sensors to get most valuable soft information back out for machine counterparts, and when/how should such soft information be obtained? These issues can be tackled within formal planning frameworks that seek to maximize the value of information (VOI) under uncertainty, so that soft data is only provided by human sensors if it is worth the cost of using limited machine and human cognitive resources to obtain it. However, exact inference and optimization for VOI-based human-machine interaction is computationally expensive, and approximations for efficient online interaction must be sought.

This study proposes a new deep learning based VOI estimation approach that aims to learn the policy reward given a joint state-action configuration. Deep learning is an emerging branch of machine learning that uses multiple levels of abstractions (from low-level features to higher-order representations, i.e., features of features) learnt from data without any hand-tuning. Among various deep learning tools, convolutional neural network (CNN) [61] is an attractive option for
extracting pertinent features from images in a hierarchical manner for detection, classification, and prediction. In addition to this deep learning architecture, other architectures such as deep relief networks, deep autoencoders, and deep recurrent neural networks have also gained immense traction as they have been shown to outperform all other state-of-the-art machine learning tools for handling very large dimensional data spaces. While most of the current applications involve image, video, speech and natural language processing, very recent studies have begun to explore its applicability for decision and control problems such as reinforcement learning [91] and guided policy search [157, 75]. In this context, although supervised learning of a state to action map may be a straightforward formulation, we show that a similar performance can be achieved while learning the reward (VOI in this case) given the state and action combination via use of a deeper architecture. We compare the performance of the deep learning based VOI estimator with a more traditional, hand-tuned policy learner such as an augmented Markov Decision process (AMDP). A feasibility study was performed with a realistic human-machine information gathering scenario for a dynamic search problem.

5.2 Human-machine collaboration

5.2.1 Problem setup

For concreteness, we will mainly focus throughout this work on information-gathering tasks involving localization of moving intruders (targets) in large environments that are incompletely covered by networks of mobile/static human and machine sensors. This serves as a useful analog for intelligence, surveillance and reconnaissance applications in the defense domain [62], as well as safety and monitoring applications. For simplicity, we focus here on localizing a single known target in a 2D environment with a known map. We also focus here on a centralized sensor network architecture, in which all machine and human sensors report observations back to a single processing point. We further assume for now that a single human sensor is tasked to provide soft data observations and is always connected to the fusion center through an appropriate interface (e.g. a workstation console or mobile device).
At discrete time step $k$, let $X_k = X \in \mathbb{R}^2$ be the unknown target location with initial prior probability distribution $p(X_0)$ at $k = 0$. Assume that the environment can be modeled as a 2D grid with $n_g$ total elements, so that $X_k$ takes on discrete realizations $x^i_k$ for $i \in \{1, \ldots, n_g\}$. A discrete time Markov chain with known transition probability $p(X_k|X_{k-1})$ is used to account for the target’s uncertain motion through the environment. The Chapman-Kolmogorov equation dictates the evolution of $p(X_{k-1})$ to $p(X_k)$,

$$p(X_k) = \sum_{x^i_{k-1}} p(X_k|X_{k-1} = x^i_{k-1}) p(X_{k-1} = x^i_{k-1}).$$

If hard sensor observations $O^h_k$ and soft sensor observations $O^s_k$ are available at each time step in the superset of observations $O_k = \{O^h_k, O^s_k\}$, then the recursive Bayes’ filter replaces $p(X_k)$ with the conditional posterior distribution $p(X_k|O_{1:k})$, where $O_{1:k} = \{O_1, \ldots, O_k\}$ and Bayes’ rule gives:

$$p(X_k|O_{1:k}) \propto p(X_k|O_{1:k-1}) p(O_k|X_k),$$

$$\propto p(X_k|O_{1:k-1}) p(O^h_k|X_k) p(O^s_k|X_k),$$

$$\propto \sum_{x^i_{k-1}} p(X_k|X_{k-1}) p(X_{k-1}|O_{1:k-1}) p(O^h_k|X_k) p(O^s_k|X_k),$$

where $O^h_k$ and $O^s_k$ are assumed conditionally independent given the true target state. The posterior summarizes all information gathered by all sensors up to time $k$ and thus efficiently updates the belief in $X_k$ without requiring storage of $O_{1:k}$. The information from each sensor is encapsulated by the observation likelihood functions $p(O^h_k|X_k)$ and $p(O^s_k|X_k)$. For hard sensors, $p(O^h_k|X_k)$ is typically derived from hardware specifications, or parametrically modeled and estimated via calibration. For soft sensors, however, $p(O^h_k|X_k)$ must be approximated as a function of $X_k$ depending on the type of human sensor interface used for a particular application. Conventional approximate Bayesian filtering techniques for non-Gaussian hard sensor data fusion (e.g. using grid, particle, or Gaussian mixture Kalman filter representations) can be naturally extended to incorporate fusion of soft data provided in various forms, including binary detection/no detection observations [3, 20] and semantic natural language observations [4, 100]. Suitable models $p(O^s_k|X_k)$ can be learned from data to properly account for uncertainty.
in human-generated information and capture key observation characteristics when deployed with real (expert or non-expert) humans.

Figure 5.1: Sample truth model simulation of 2D target localization problem, showing locations of cameras 1-6 and associated fields of views, along with posterior distribution \( p(X_k | O_{1:k}) \) (heat map) and true target location (magenta x) for a random walk motion model \( p(X_{k+1} | X_k) \). Each sensor has 0.99 detection rate and false alarm rate of \( 1 \times 10^{-4}, 0.1504, 0.1585, 0.1992, 0.1510, \) and 0.1593, respectively.

The sensor scheduling problem in this context thus becomes one of obtaining an appropriate set of hard/soft measurements in the superset \( O_{1:k} \) so that the posterior distribution remains as informative as possible for constrained intelligent decision-making (e.g. deploying security assets to intercept the intruder within a certain time window). For example, consider the 2D grid world shown in Fig. 5.1, which features 6 cameras located in large open environment with a moving target that obeys random walk dynamics. Some of the cameras may be linked to automatic target recognition (ATR) algorithms, with known true detection and false alarm rates that define \( p(O^h_k | X_k) \). Alternatively, a remote human analyst can also access each camera in order to declare whether or not the target is in the camera’s field of view (independently of ATR software). Due to the varying quality of each camera and various cognitive loads, the human analyst may incorrectly declare that the target has (not) been detected at any given camera with some true detection and false alarm rates defining \( p(O^s_k | X_k) \). In either case, we assume the data fusion center can only request one hard observation \( O^h_k \) or soft observation \( O^s_k \) at any given \( k \) from a single camera, due to bandwidth and processing restrictions. The scheduling problem thus addresses the question of which single har or soft sensor to query for the Bayes filter update at time \( k \). This leads to the more general problem of obtaining an optimal sequence.
of hard/soft sensor queries to maximize some utility function over time. Matters become even more interesting and challenging when we consider that human sensors can also provide detailed semantic observations, e.g., ‘Target is now headed north towards the door very quickly’. In this case, given a known dictionary of grounded semantic statements that can be translated into target state information, the data fusion center could also schedule and execute the most informative semantic human sensor queries over time, which could be answered in free-form (e.g. ‘Tell me what you see in camera 2?’) or binary manner (‘Is the target moving to the door in camera 2?’).

5.2.2 Value of information (VOI) for soft/hard sensor scheduling

Previous work on combined soft/hard sensing focused primarily on instances where human sensors voluntarily ‘push’ useful information as they see fit. However, this can lead to suboptimal gains for machine sensing and planning performance, especially if the human analyst becomes distracted or fails to recognize when machine sensors are unable to collect good data. This leads us to consider formal strategies for opportunistically ‘pulling’ information from human sensors in the right way at the right time to enhance state estimation and long-term decision-making under uncertainty. Such interactions should also account for the costs of interacting with human sensors, in order to help manage their limited cognitive resources and avoid task overloading.

These issues are naturally addressed via formal decision-theoretic Bayesian inference to assess the value of information (VOI) for different soft data reports [59]. For the simple target localization problem, suppose $O_k^s \in \{o_k^{s,1}, \ldots, o_k^{s,n_s}\}$, where $o_k^{s,j} \in [0,1]$ denotes a specific kind of binary soft observation report. For instance, $o_k^{s,j}$ could represent a detection/no detection event for camera $j$, or a binary true/false response to a semantic query $j$ from a large list of possible queries.
Given a utility function \( U(D_k, X_k) \) representing the expected long-term benefit of taking some discrete action \( D_k \) while the target is in state \( X_k \), the VOI for receiving a single noisy report \( o_{k}^{s,j} \) in response to a soft data query is:

\[
\text{VOI}(o_{k}^{s,j}) = \mathbb{E} \left[ \max_{D_k} U(D_k, X_k) \right] - \max_{D_k} \mathbb{E} [U(D_k, X_k)](X_k)
\]

where \( \mathbb{E}[f](\nu) \) is the expected value of \( f \) over random variables \( \nu \), and:

\[
\mathbb{E} \left[ \max_{D_k} U(D_k, X_k) \right]_{o_{k}^{s,j}, X_k} = \sum_{o_{k}^{s,j}} p(o_{k}^{s,j}|O_{1:k-1}) \left[ \max_{D_k} \sum_{x_k} p(X_k|O_{1:k-1}, o_{k}^{s,j}) U(D_k, X_k) \right]
\]

Assuming a cost \( c(o_{k}^{s,j}) \) for obtaining \( o_{k}^{s,j} \), then the human sensor should be queried for \( o_{k}^{s,j} \) if \( \text{VOI}(o_{k}^{s,j}) > c(o_{k}^{s,j}) \). Thus, (5.1) gives a formal way to assess whether the expected information from report \( o_{k}^{s,j} \) is worth the cost of retrieving it, regardless of the outcome of \( o_{k}^{s,j} \). The key idea in eq. (5.1) is to compare the maximum expected utility given all possible outcomes for \( X_k \) and \( o_{k}^{s,j} \) to the maximum expected utility if no new soft data were obtained. In practical applications, we must compare the VOI at time \( k \) for \( n_s \) alternative sensor queries \( o_{k}^{s,j}, j \in \{1, ..., n_s\} \), and select only the query with the highest VOI. This is referred to as a myopic approximation, since it does not consider all possible combinations of observations \( o_{k}^{s,j} \) that could be taken together at time \( k \). For dynamical systems, this approach is also myopic in the sense that it does not consider future sensing actions for time \( k+1 \) and beyond. However, the definition of VOI can be generalized to find an optimal non-myopic soft querying sequence \( O_{k:k+T}^s \) for \( T > 0 \), by assessing the single best query \( O_k^s \) at each time step \( k, ..., k+T \), and comparing the final expected utility at step \( k+T \) to the expected utility with respect to \( p(X_{k+T}|X_k, O_{1:k}) \) (the propagated belief without any future soft observations).

VOI depends heavily on \( U(D_k, X_k) \) and \( c(o_{k}^{s,j}) \), as well as the uncertainty in \( p(X_k|O_{1:k}) \). For human sensors, \( c(o_{k}^{s,j}) \) can be related to the expected cognitive cost of re-tasking human sensors [59]. As there is no standard way to define \( c(o_{k}^{s,j}) \) for general applications, for simplicity and without loss of generality, we ignore \( c(o_{k}^{s,j}) \) for now and only consider utility defined by expected information gain for sensing actions. In this case, \( D_k \) is related only to the choice of
$j \in \{1, ..., n_s\}$ and we seek to minimize the entropy of $p(X_k|O_{1:k-1}, o^{s,j}_k)$, so that the expression becomes:

$$U(o^{s,j}_k, X_k) = \log p(X_k|O_{1:k-1}, o^{s,j}_k). \quad (5.1)$$

Hence, the VOI for $o^{s,j}_k$ in (5.1) becomes the expected decrease in posterior entropy,

$$\text{VOI}(o^{s,j}_k) = \mathbb{E}[\mathcal{H}[p(x_k|O_{1:k-1}, o^{s,j}_k)]]_{(o^{s,j}_k)} - \mathcal{H}[p(x_k|O_{1:k-1})];$$

where $\mathcal{H}[p(x_k|O_{1:k})] = \mathbb{E}[\log p(x_k|O_{1:k-1})](x_k)$

This means that soft data $o^{s,j}_k$ will be (myopically) requested to keep the overall spread of uncertainty in $p(x_k|O_{1:k})$ as small as possible. Entropy minimization is also widely used for tasking of hard sensors in target localization applications [54], and thus provides a useful common objective for combined hard-soft sensor scheduling.

### 5.2.2.1 Practical VOI inference and optimization

Although VOI is an ideal measure by which to formally regulate human-machine interaction, VOI calculations are computationally expensive and lead to NP-hard Bayesian inference calculations for marginal observation likelihoods $p(o^{s,j}_k|O_{1:k-1})$. The comparison of VOI for various $o^{s,j}_k$ reports can also be quite expensive when $n_s$ is large (e.g. for large semantic dictionaries), or if temporally non-myopic query sequences over multiple time steps are considered (due to combinatorial blow up). Hence, even for myopic approximations, it is generally impractical to explicitly compute and compare the VOI among $n_s$ soft sensing alternatives. While many approximate inference methods have been developed to address these issues [81], these are still computationally expensive to implement as they still require online inference and optimization.

It is worth noting that VOI-based sensor scheduling problems can also be interpreted as partially observable Markov decision processes (POMDPs) [64], which advantageously allow \textit{optimal sensing policies} $\pi(b(X_{k-1}))$ to be computed offline via value iteration over the belief space $b(X_{k-1}) = p(X_k|O_{1:k-1})$. Such policies represent direct ‘look-up tables’ from $b(X_{k-1})$ to sensing queries $o^{s,j}_k$ that automatically account for VOI through expected cumulative rewards, and thus allow for efficient online optimal querying that bypasses explicit computation and comparison of
all possible future sensing actions. Although there are many well-known approximate techniques for solving standard POMDPs with linear additive reward functions over $X_k$, information-based utilities such as negative entropy lead to non-standard POMDPs, since the expected rewards are non-linear functions of $b(X_{k-1})$. Hence, many state-of-the-art approximate POMDP solvers cannot be directly applied here. However, as discussed in Section 4, approximate methods such as the augmented Markov decision processes (AMDPs) [113, 142] can be used to derive sensing policies by first learning and then solving MDPs over belief space features $f(b(X_k))$. AMDPs recover the linear additive reward structures in $f(b(X_k))$ space and thus enable the use of standard value iteration solutions for offline approximate policy generation for non-standard POMDPs. This is possible due to the fact that $b(k)$ is the sufficient statistic for POMDP policy calculations. However, it is non-trivial to define the features $f(b(k))$ which re-produce the optimal total expected rewards for the original POMDP. Furthermore, the corresponding MDP model parameters must be obtained via simulation for offline policy calculation, subject to hand-tuning of discount and immediate expected reward parameters.

Nevertheless, the AMDP approximation for POMDPs provides insight into other possible approximation strategies for optimal VOI-based sensor scheduling. In particular, it suggests that optimal querying policy maps could be obtained offline via supervised learning, using features of $b(k)$ and simulated observation instances to provide generalizable VOI associations between $b(k)$ and $O_k^s$ queries. The resulting learned ‘look up table’ function could be trained for non-myopic querying, and would thus provide a computationally efficient means for pulling information from soft sensors that avoids expensive online brute force VOI optimization over all possible query sequences.

### 5.3 Deep VOI estimators

Deep neural networks present such an attractive option for estimating VOI without policy hand-tuning. Specifically for the purpose of the study, CNN is a suitable choice due to its ease of training while still achieving a comparable performance despite the fact that it has fewer parameters relative to other fully connected networks with the same number of hidden layers. Furthermore, CNNs are designed to exploit the 2D structure of an input image (images of belief
space in this case) by preserving the locality of features via the utilization of spatially-local correlations of an image through the use of tied weights, therefore being invariant to translations.

5.3.1 Design of framework

**Policy Learning:** A typical formulation of the problem is directly mapping the belief space to the action by using belief maps as inputs and sensor indices as outputs. For this problem set-up, a single belief map is accompanied by an index denoting the action of switching into the sensor that gives the best VOI gain. The belief map is represented by a $15 \times 18$ (rows $\times$ columns, in pixels) grayscale image, where pixels with high grayscale intensity denote high belief and pixels with low grayscale intensity denote low belief. The belief map is then vectorized into a vector of $1 \times 270$ units and goes through a typical convolutional neural network to make a prediction that suggests which sensor to switch into for the most VOI gain.

Another formulation of the problem that is perhaps more robust, we aim to learn the underlying function for predicting VOIs given a belief map and an arbitrary sensor index. For this formulation, the CNN architecture is similar but the inputs are modified to include an arbitrary sensor. The Cartesian coordinates of this sensor is indicated on a map with the same dimensions of the belief map called the **sensor map**. In addition, enough padding is added between the belief map and the sensor map such that the information from these two spaces do not interfere with the learning of filter weights. The belief map ($15 \times 18$), padding ($10 \times 18$), and sensor map ($15 \times 18$) are vertically concatenated to form a $40 \times 18$ image that will become
the input to the CNN. VOI also becomes the true class labels associated with this input image during training. However, since VOI is real-valued, we have discretized the VOI space into 62 classes to formulate a classification problem. Hence, the outputs of the model (i.e., the classes) correspond to the estimated VOI given a sensor and a belief map. Fig. 5.2 shows a schematic of this formulation. The VOI can then be estimated based on the associated action of switching into a particular sensor given the current belief map.

**Training Data Generation:** Data is generated from running the simulation for 10,000 time steps with random sensor false alarm rate of $15 \pm 5\%$ and the target moving in a random walk manner. Depending on the formulation of the problem, the resultant size of the dataset varies. In the policy learning formulation, we are limited to one training example per time step since there is only one optimal sensor index given a single belief map. This results in 10,000 examples, where half of them (i.e., 5,000) is used for training and the other half is used for cross-validation to avoid overfitting. The reward learning formulation allows us to produce six examples per time step (since there are six sensors in the field) and generated a total of 60,000 examples. In this case, we consider the expectation of VOI gained 4 time steps ahead in future, in a non-myopic fashion. The expected VOI gained by switching into the particular sensor is used as the class labels after discretization. Again, half of the example data is used for training and the other half is used for validation.

**Network Architecture and Hyper-parameters:** Specific details on the number of filters, convolutional layer, pooling layers, and fully connected layers are shown in and Fig. 5.2. A learning rate of 0.01 is used for the gradient descent algorithm for training the CNN in a supervised manner with a batch size of 10 samples. The optimized model is acquired prior to the point when validation error becomes consistently higher than the training error in subsequent training iterations.
5.4 Results and discussion

In this section, the performance of CNN model is evaluated and compared with AMDP.

5.4.1 Prediction accuracy

Figure 5.3: Four examples of belief map and VOI (discretized into bins) associated with the action of switching into a particular sensor at different simulation time steps $T$.

Figure 5.4: Estimated VOI vs. True VOI for the first 40 out of 10,000 simulation time steps. The figure also shows whether the sensor selected through rank-ordering the predicted VOI is the same as the sensor selected by ordering the true VOI, represented by green circles (matching) and red crosses (non-matching).
The VOI estimated by the trained model follows closely to the true VOI computed as shown in Fig. 5.4. Each point in one time step partition represents a sensor index; the model has to select one sensor out of six based on the current belief map with the highest expected VOI gain. A successful selection is represented by a green circle, whereas a wrong selection is represented by a red cross. It is observed that CNN can be in fact trained to learn the reward function and produce VOIs similar to the values computed by the brute force simulation. Most importantly, the high capability of predicting this non-myopic VOI suggests that the model does not naively generate VOI that is directly proportional to the grayscale pixel intensity of the belief map (which corresponds to the probability of finding the target) at that particular sensor location. Furthermore, careful inspection of the camera choice errors reveal that most of the errors come from confusing between camera 5 and camera 6 that are spatially very close as well as tend to have similar VOI reward for many belief space configurations.

Results also suggest that the trained CNN model is able to generalize into unseen belief maps and still predict VOI that is close to the true VOI. To show this, we have reduced the size of the dataset used to train the CNN model with $p \in \{0.4, 0.6, 0.8, 1.0\}$ denoting the fraction of the new training and validation set sizes from their original sizes. From Fig. 5.5, the accuracy of selecting the best sensor with largest VOI gain decreases slightly with smaller dataset sizes. Policy learning formulation (i.e., direct action mapping) seems to perform slightly better compared to reward learning (i.e., mapping into VOI) as it suffers only a slight degradation in prediction performance. Furthermore, policy learning is attractive because it is straightforward and intuitive - given a belief map, a decision can be made to choose the best sensor. Note, the reward learning
framework needs a deeper architecture (i.e., an extra fully connected layer) to achieve a similar performance compared to the policy learning framework. Training the CNN for policy is also a little easier as it allows a simpler model with the number of output classes equivalent to the number of available options (i.e., choosing a particular sensor). On the other hand, the ability to evaluate the VOI from an action carried out in a specific belief space is very useful in situations where the location/characteristics of the sensors may be altered slightly. Additionally, the framework can be more flexible in terms of additional objectives or imposing extra constraints (e.g., the predicted VOI can be processed to include penalization terms for certain actions). If this happens, the policy learning model must be retrained whereas the reward learning model may not need to be retrained. The adaptability of the reward learning formulation is absolutely valuable for generalizing into larger problem setups by paying a small price in accuracy.

5.4.2 Comparison with AMDP

A policy derived from a feature-based solution to the POMDP model for the sensor scheduling problem can be used to provide a baseline comparison with the learned CNN policies. As mentioned in Section 2, augmented Markov Decision processes (AMDPs) can be used to solve the non-standard POMDP for sensor scheduling when the reward function is defined to minimize the entropy of the posterior state belief $b_k = p(X_k|O_{1:k})$. AMDPs use a learned Markov decision process (MDP) model on features $f(b_k)$ of the belief space, which can include the entropy of $b_k$. This MDP can then be used to derive scheduling policies via offline value iteration with linear additive rewards defined in terms of $f(b_k)$ instead of $X_k$. By choosing a good set of features, the optimal expected total reward for the AMDP policy can closely match that of the original POMDP defined over $X_k$ with non-standard entropy rewards. As discussed in [113, 142], the key idea behind the AMDP is that most of the reachable belief space $b_k$ evolves along a low-dimensional manifold, which can be encoded by a feature set whose size is typically much smaller than the number of possible states $X_k$ in the original POMDP. Hence, AMDPs offer a generative feature-based alternative to finding scheduling policies, in contrast to the discriminative feature-based modeling approach used by CNNs.
However, the problem remains to find suitable features \( f(b_k) \) and learn the corresponding MDP model over the feature space, which is a non-trivial learning problem. Furthermore, discount and immediate expected reward parameters for the AMDP must be hand-tuned to arrive at suitable policies via value iteration. Finally, stationary infinite horizon policies must often be used in practice to avoid the computational cost of performing finite time value iterations for non-stationary scheduling policies (especially if discount and reward parameters must be tuned). The use of infinite horizon policies necessarily makes the AMDP suboptimal for finite-horizon querying problems, but nevertheless provides insight into the expected capabilities of generative feature-based learning approaches for sensor scheduling policy estimation. Our implementation of AMDP for the simple 2D grid world problem follows the basic technique presented in [142], which defines \( f(b_k) \) as the stacked vector of the maximum a posteriori (MAP) state \( X_k \) of \( b(k) \) and the (discretized) entropy of \( b(k) \). The resulting implementation used 27,000 AMDP feature states, and the policy was generated using hand-tuned rewards (with positive rewards proportional to inverse square of entropy for transitions to lower-entropy states, and negative rewards otherwise) and a fixed value function discount factor of 0.1 (to encourage earlier transitions to low-entropy states).

![Figure 5.6: Entropy of belief map over time.](image)

To study the level of certainty where the target is in a particular region, the entropy of the belief map at each time step is studied. High entropy means there is a large uncertainty that the target is in a given region, and low entropy suggests an ability of narrowing down the target
location. The entropy of the belief map generated using AMDP and CNN is computed and plotted over 500 time steps, shown in Fig. 5.6. Belief entropy from AMDP experiences sharp dips in magnitude occasionally but stays high at all other times. Entropy from CNN is generally lower at most time steps, thus implying lower uncertainty. However, this may not be so simple to say, especially given the target’s propensity to move around a lot—even if the model starts off knowing exactly where the target is, the entropy jumps up a lot at the next time step. Therefore, along with this decision uncertainty metric, a correctness metric is also used to compare the performances of AMDP and CNN.

Figure 5.7: Euclidean error between MAP estimate and target location over time.

Figure 5.8: Error distribution between MAP estimate and target location.

Estimation error is defined as the Euclidean distance between the maximum a posteriori probability (MAP) estimate (i.e., largest value of the belief map) and the actual target location. Ideally, this error should be as close to zero as possible. The error is calculated using belief map and target locations for each time step (see Fig. 5.7) and its distribution is shown in Fig. 5.8.
Clearly, CNN outperforms AMDP as the distribution is more skewed with higher probability of the belief map mode being closer to the actual target location, hence increasing the success rate of target-tracking.

**Remark 5.4.1** As a hierarchical feature extraction tool, deep CNN is able to extract belief space features at different spatial scales as well as obtain ‘features of features’. Also, belief spaces generally evolve on low-dimensional manifolds (i.e., feature spaces) as suggested by the AMDP formulation. Therefore, it becomes feasible for a deep CNN to associate these representative features to the VOI. In this context, we make an interesting observation (as shown in Fig. 5.5) that CNN performance does not degrade much with a drastic decrease in the training data size. This probably suggests that there is a relatively low number of key features in the belief space that (the CNN is able to learn) lead to various belief space configurations via complex combinations. Another view could be that the proposed framework aims to automatically learn the probability density of the reward given belief space and action. Therefore, with a hierarchy of nonlinear functions and a large number of model parameters, it becomes feasible to model arbitrarily complex densities. However, sufficient training data and regularization steps are necessary to avoid overfitting.

### 5.5 Semantic soft data scheduling

Consider a ‘cops and robbers’ scenario in which one robot (the ‘cop’) searches for another mobile intruder robot (the ‘robber’) in an indoor environment, with a simulated remote human ‘security guard’ providing observations of the robber’s 2-dimensional position state. As in the previous grid world toy problem, the cop maintains a Bayesian belief map over the environment of the robber’s state, and updates this belief either through fusion of hard sensor measurements (e.g. detection of robber position via an on-board camera) or soft data. The soft data in this case takes the form of a human semantic observation as in [4], e.g. “The robber is in front of the desk.” We assume that the remote human views the scene either through the cop robot’s camera, or a security camera placed in each room. The human thus has full visibility into the space, but is constrained to visibility of no more than one room per time step. Similarly, we
have constrained the cop robot to asking a single question about the robber once every $n_q = 5$

time steps, as a way to mitigate operator load. Over time, the probability mass of the target

position diffuses over the environment, based on the cop’s estimate of the robber’s position and

known random-walk dynamics model.

With this setup, the human observations $o_k^{s,j}$ can be thought as binary ‘true/false’ responses
to the questions asked by the cop robot, where $j$ indexes an element from a finite list of $n_s$
semantic questions. We assume $o_k^{s,j}$ arrive only as responses to questions posed by the robot, i.e.
the human does not ‘push’ information voluntarily, and only one semantic query $j$ is selected
from $\{1,...,n_s\}$. The cop can ask the human whether the robber is either inside one of the
rooms or near one of the objects shown in Fig. 5.9. This leads to $n_s = 16$ possible questions
that generate measurement updates. For example, if the cop asks, "Is the robber near the dining
table?", a positive response triggers fusion of the likelihood shown in Fig. 5.10 with the current
belief state. At any given query instance, the cop seeks to ask the question with maximum VOI,
as defined by the expected reduction in entropy. The $16^T$ queries of depth $T$ are ranked by their
corresponding VOI; in the non-myopic case of $T \geq 2$, all question paths are ranked by VOI first
and then reduced to the initial 16 possible questions, ranked by the maximum VOI of all query
sequences starting with that question.

Figure 5.9: The indoor search environment and associated semantic features for soft observations.

**CNN training:** This scenario provides a truth model used to train the CNN. Applying

the same CNN input-output structure and problem formulation, the belief map ($72 \times 136$) is
Figure 5.10: The likelihood function that maps to the example observation, ‘Target is near the dining table’.

appended with padding (10 × 136) and the action map (8 × 136) where the action map is divided into 16 equally spaced nodes that light up depending on the question posed by the cop. Enough padding is created to separate the belief map from the action map for efficient learning of the convolving filters. Fig. 5.11 illustrates the images used in learning. We trained the CNN model (only for myopic scenario for this feasibility study) with 31,680 training examples and 31,680 validation examples sampled from 100 simulation trials with 400 time steps each. Therefore, variability in the belief space is sufficient. As the input image size is now larger relative to the simpler problem, filter sizes are increased to 10 of 7 × 9 in the first convolutional layer and 25 of 5 × 5 in the second convolutional layer with all other parameters equal. Note that the length of the smaller padding edge (i.e., 10 pixels) is still larger than the longest filter dimensions.

**Results:** From our simulations, the CNN is **91.33%** accurate in determining the next best question with highest VOI gain that should be posed by the robot (based on 1,980 time steps), as compared with the ground truth determined by brute force calculation. Interestingly, most of the errors occur as the target moves from one room to a neighboring one (e.g., from library to study) and the query selection sometimes remain associated with the previous room. The CNN-based 'query' selection quickly adjusts within a few time steps after such a transition. Hence, most of these errors can potentially be avoided with a non-myopic VOI implementation for this real-life scenario and is currently being pursued. Overall, this exercise demonstrates the feasibility of using deep learning architectures for such decision-making processes.
Figure 5.11: Nine examples of CNN input showing the belief map, padding, and the action map in the realistic scenario.

5.6 Conclusions and future work

VOI-based human-machine interfaces can automatically determine how and when to present queries to human operators in a real problem. However, practical online implementation of VOI-based querying strategies remains challenging, since the problem of selecting the optimal sequence of queries leads to a difficult analytically intractable joint optimization and inference problem. This study uses recent advancements in deep learning to build a VOI estimation framework that is shown to be able to reliably estimate VOI without any policy hand-tuning. A 2-D grid world search problem (with a moving target) is used to compare the performance of the finite horizon deep VOI estimator with that of a hand-tuned AMDP policy. Simulation results show that a CNN-in-the-loop information gathering system is able to lower the expected entropy of belief spaces and the expected error between the MAP estimate of the target and the true target compared to an AMDP-in-the-loop process. Finally, a feasibility study was performed on a simulation test bed with a realistic human-machine collaboration problem.

Better network design and hyper-parameter optimization are currently being investigated. Other future research directions are: (i) online tuning of deep networks; (ii) hybrid approaches involving deep feature extractors and traditional planning algorithms (e.g. using CNNs to learn feature maps for AMDP-based policy approximations); (iii) addressing sensor modeling issues,
including potentially unknown false alarms/missed detection rates and imperfect human sensor models (e.g. using hierarchical Bayesian modeling as in [3] to account for model uncertainties); and (iv) validation of the proposed methodology on indoor robotic target search test bed with live human users. We will also extend the comparisons made here between feature-based direct policy learning approaches and feature-based POMDP approximations to other state-of-the-art POMDP approximations, including those that approximate low-dimensional reachable belief spaces via online sampling rather than through offline-learned feature compression [129].
CHAPTER 6. LLNET: A DEEP AUTOENCODER APPROACH TO NATURAL LOW-LIGHT IMAGE ENHANCEMENT

In surveillance, monitoring and tactical reconnaissance, gathering visual information from a dynamic environment and accurately processing such data are essential to making informed decisions and ensuring the success of a mission. Camera sensors are often cost-limited to capture clear images or videos taken in a poorly-lit environment. Many applications aim to enhance brightness, contrast and reduce noise content from the images in an on-board real-time manner. We propose a deep autoencoder-based approach to identify signal features from low-light images and adaptively brighten images without over-amplifying/saturating the lighter parts in images with a high dynamic range. We show that a variant of the stacked-sparse denoising autoencoder can learn from synthetically darkened and noise-added training examples to adaptively enhance images taken from natural low-light environment and/or are hardware-degraded. Results show significant credibility of the approach both visually and by quantitative comparison with various techniques.

6.1 Motivation

Good quality images and videos are key to critical automated and human-level decision-making for tasks ranging from security applications, military missions, path planning to medical diagnostics and commercial recommender systems. Clean, high-definition pictures captured by sophisticated camera systems provide better evidence for a well-informed course of action. However, cost constraints often limit large scale applications of such systems. Thus, relatively inexpensive sensors are used in many cases. Furthermore, adverse conditions such as insufficient lighting (e.g. low-light environments, night time) worsen the situation. As a result, many
areas of application, such as Intelligence, Surveillance and Reconnaissance (ISR) missions (e.g. recognizing and distinguishing enemy warships), unmanned vehicles (e.g. automated landing zones for UAVs), and commercial industries (e.g. property security, personal mobile devices) stand to benefit from improvements in image enhancement algorithms.

While neural networks have been widely studied for image denoising tasks, we are not aware of any existing works using deep networks to both enhance and denoise images taken in poorly-lit environments. In the present work, we approach the problem of contrast enhancement from a representation learning perspective using deep autoencoders (what we refer to as Low-light Net, or LLNet) that are trained to learn underlying signal features in low-light images and adaptively brighten and denoise. The method takes advantage of the local patch-wise contrast improvement similar to the works in [87] to enhance contrast such that the improvements are done relative to local neighbors to prevent over-amplifying the intensities of already brightened pixels. Furthermore, the same neural network is trained to learn the structures of objects that persist through noise in order to produce a brighter, denoised image.

**Contributions:** The present paper presents a novel application of using a class of deep neural networks–stacked sparse denoising autoencoder (SSDA)–to enhance natural low-light images. To the best of the author’s knowledge, this is the first application of using a deep architecture for (natural) low-light image enhancement. We propose a training data generation method by synthetically modifying images available on Internet databases to simulate low-light environments. Two types of deep architecture are explored - (i) for simultaneous learning of contrast enhancement and denoising (LLNet) and (ii) sequential learning of contrast-enhancement and denoising using two modules (staged LLNet or S-LLNet). The performances of the trained networks are evaluated and compared against other methods on test data with synthetic noise and artificial darkening. Performance evaluation is repeated on natural low-light images to demonstrate the enhancement capability of the synthetically trained model applied on a realistic set of images obtained with regular cell-phone camera in low-light environments. Hidden layer weights of the deep network are visualized to offer insights to the features learned by the model. Another contribution is that the framework performs *blind* contrast enhancement without requiring a reference image frame (e.g. using information from a previous frame in
video enhancement [31], and the use of daytime counterparts [153]), which is absolutely vital in scenarios where new environments are frequently encountered (e.g. in tactical reconnaissance).

6.2 Related work

There are well-known contrast enhancement methods such as improving image contrast by histogram equalization (HE) and its variants such as contrast-limiting adaptive HE (CLAHE) brightness preserving bi-HE (BBHE) and quantized bi-HE (QBHE) [103, 102, 66, 60]. Subsequently, an optimization technique, OCTM [150] was introduced for mapping the contrast-tone of an image with the use of mathematical transfer function. However, this requires weighting of some domain knowledge as well as an associated complexity increase. Available schemes also explored using non-linear functions like the gamma function [42] to enhance image contrast.

Image denoising tasks have been explored using BM3D [28], K-SVD [34], and non-linear filters [24]. Using deep learning, authors in [145] presented the concept of denoising autoencoders for learning features from noisy images while [57] applied convolutional neural networks to denoise natural images. In addition, authors in [2] implemented an adaptive multi-column architecture to robustly denoise images by training the model with various types of noise and testing on images with arbitrary noise levels and types. Stacked denoising autoencoders were used in [22] to reconstruct clean images from noisy images by exploiting the encoding layer of the multilayer perceptron (MLP).

Fotiadou et al. [39] enhanced natural low-illumination images using sparse representations of low-light image patches in an appropriate dictionary to approximate the corresponding daytime images. Dong et al. [31] proposed an algorithm that inverts the dark input frames and performs de-hazing to improve the quality of the low light images. A related method is presented in [158] involving de-hazing algorithms. Another technique, proposed in [153], separates the image into two components—reflectance and illuminance—and enhance the images using the reflectance component. The separation of the components, however, is difficult; therefore it may introduce unwanted artifacts in the reconstructed images.

Perhaps one of the most challenging tasks is to gather sufficiently large dataset of low-light images to train the deep learning model. The NORB object recognition dataset [73] contains
natural images taken at 6 different illumination levels, but the limited size of the training set is insufficient for training. With this motivation, we also propose a method of simulating low-light environments by modifying images obtained from existing databases.

6.3 The Low-light Net (LLNet)

The proposed framework is introduced in this section along with training methodology and network parameters.

6.3.1 Learning features from low-light images with LLNet

SSDAs are sparsity-inducing variant of deep autoencoders that ensures learning the invariant features embedded in the proper dimensional space of the dataset in an unsupervised manner. Early proponents [145] have shown that by stacking several denoising autoencoders (DA) in a greedy layer-wise manner for pre-training, the network is able to find a better parameter space during error back-propagation. The mechanics of stacked denoising autoencoders is outlined in section 2.1.4.

LLNet framework takes its inspiration from SSDA whose sparsity-inducing characteristic aids learning features to denoise signals. In the present work, we take the advantage of SSDA’s denoising capability and the deep network’s complex modeling capacity to learn features underlying in low-light images and produce enhanced images with minimal noise and improved contrast. A key aspect to be highlighted is that the network is trained using images obtained from internet databases that are subsequently synthetically processed (i.e. darkening nonlinearly and adding Gaussian noise) to simulate low-light conditions, since collection of a large number of natural low-light images (sufficient for deep network training) and their well-lit counterparts can be unrealistic for practical use. Despite the fact that LLNet is trained on synthetic images, both synthetic and natural images are used to evaluate the network’s performance in denoising and contrast-enhancement.

Aside from the regular LLNet where the network is trained with both darkened and noisy images, we also propose the staged LLNet (S-LLNet) which consists of separate modules arranged in series for contrast enhancement (stage 1) and denoising (stage 2). The key distinction over
Figure 6.1: Architecture of the proposed framework: (a) An autoencoder module is comprised of multiple layers of hidden units, where the encoder is trained by unsupervised learning, the decoder weights are transposed from the encoder and subsequently fine-tuned by error back-propagation; (b) LLNet with a simultaneous contrast-enhancement and denoising module; (c) S-LLNet with sequential contrast-enhancement and denoising modules. The purpose of denoising is to remove noise artifacts often accompanying contrast enhancement.

the regular LLNet is that the modules are trained separately with darkened-only training sets and noisy-only training sets. Both structures are presented in Fig. 6.1. Note, while the S-LLNet architecture provides a greater flexibility of training, it increases the inference time slightly which may be a concern for certain real-time applications. However, customized hardware-acceleration can resolve such issues significantly.

6.3.2 Network parameters

LLNet is comprised of 3 DA layers, with the first DA layer taking the input image of dimensions $17 \times 17$ pixels (i.e. 289 input units). The first DA layer has 2,000 hidden units, the second has 1,600 hidden units, and the third has 1,200 hidden units which becomes the
bottleneck layer. Beyond the third DA layer forms the decoding counterparts of the first three layers, thus having 1,600 and 2,000 hidden units for the fourth and fifth layers respectively. Output units have the same dimension as the input, i.e. 289. The network is pre-trained for 30 epochs with pre-training learning rates of 0.1 for the first two DA layers and 0.01 for the last DA layer, whereas finetuning was performed with a learning rate of 0.1 for the first 200 finetuning epochs, 0.01 afterwards, and stops only if the improvement in validation error is less than 0.5%. For the case of S-LLNet, the parameters of each module are identical.

6.3.3 Training data generation

Training was performed using 422,500 patches, extracted from 169 standard test images\(^1\). Consistent with current practices, the only pre-processing done was to normalize the image pixels to between zero and one. During the generation of the patches, we produced 2,500 patches from random locations (and with random darkening and noise parameters) from the same image. Note that the images used for generating patches for the training set and the validation set are disjoint in order to reduce the correlation between the training and validation set. By doing so, we avoid correlation between the two sets which has the potential to overestimate the model performance. The $17 \times 17$ pixel patches are then darkened nonlinearly using the MATLAB command \texttt{imadjust} to randomly apply a gamma adjustment. Gamma correction is a simple but general case with application of a power law formula to images for pixel-wise enhancement with the following expression:

$$I_{\text{out}} = A \times I_{\text{in}}^\gamma$$

where $A$ is a constant determined by the maximum pixel intensity in the image. Intuitively, image is brightened when $\gamma < 1$ while $\gamma = 1$ leaves it unaffected. Therefore, when $\gamma > 1$, the mapping is weighted toward lower (darker) grayscale pixel intensity values.

6.3.4 Simulating darkness

A uniform distribution of $\gamma \sim \text{Uniform} (2, 5)$ with random variable $\gamma$ is selected to result in training patches that are darkened to a varying degree. To simulate low quality cameras used to

\(^{1}\text{Dataset URL: http://decsai.ugr.es/cvg/dbimagenes/}\)
capture images, these original training patches are corrupted by Gaussian noise via the MATLAB function `imnoise` with standard deviation of $\sigma = \sqrt{B(25/255)^2}$, where $B \sim \text{Uniform } (0, 1)$. Hence, the final corrupted image and the original image exhibit the following relationship:

$$I_{\text{train}} = n(g(I_{\text{original}})) \quad (6.2)$$

where function $g(\cdot)$ represents the gamma adjustment function and $n(\cdot)$ represents the noise function.

Random gamma darkening with random noise levels result in a variety of training images that can increase the robustness of the model. In reality, natural low-light images may also include quantization and Poisson noise (e.g. images captured with imaging sensors such as CCD and CMOS) in addition to Gaussian noise. We chose to focus on the Gaussian-only model for the ease of analysis and as a preliminary feasibility study of the framework trained on synthetic images and applied to natural images. Furthermore, since Gaussian noise is a very familiar yet popular noise model for many image denoising tasks, we can acquire a sense of how well LLNet performs with respect to other image enhancement algorithms. The training set is divided into 211,250 training examples, 211,250 validation samples, and the samples are subsequently randomly shuffled. The training step involves learning the invariant representation of low light and noise with the autoencoder described in section 6.3.2. While training the model, the network attempts to remove the noise and simultaneously enhance the contrast of these darkened patches. The reconstructed image is compared against the clean version (i.e. bright, noiseless image) by computing the mean-squared error.

When training both LLNet and S-LLNet, each DA is trained by error back-propagation to minimize the sparsity regularized reconstruction loss as described in Xie et al. [151]:

$$\mathcal{L}_{\text{DA}}(D; \theta) = \frac{1}{N} \sum_{i=1}^{N} \frac{1}{2} || y_i - \hat{y}(x_i) ||_2^2 + \beta \sum_{j=1}^{K} \text{KL}(\hat{\rho}_j || \rho) + \frac{\lambda}{2} (||W||_F^2 + ||W'||_F^2) \quad (6.3)$$

where $N$ is the number of patches, $\theta = \{W, b, W', b'\}$ are the parameters of the model, $\text{KL}(\hat{\rho}_j || \rho)$ is the Kullback-Leibler divergence between $\rho$ (target activation) and $\hat{\rho}_j$ (empirical average activation of the $j$-th hidden unit) which induces sparsity in the hidden layers:

$$\text{KL}(\hat{\rho}_j || \rho) = \rho \log \frac{\rho}{\hat{\rho}_j} + (1 - \rho) \log \frac{1 - \rho}{1 - \hat{\rho}_j} \quad \text{where} \quad \hat{\rho}_j = \frac{1}{N} \sum_{i=1}^{N} h_j(x_i) \quad (6.4)$$
and $\lambda$, $\beta$ and $\rho$ are scalar hyper-parameters determined by cross-validation. After the weights of the decoder have been initialized, the entire pre-trained network is finetuned using an error back-propagation algorithm to minimize the loss function given by:

$$
\mathcal{L}_{SSDA}(D; \theta) = \frac{1}{N} \sum_{i=1}^{N} ||y_i - \hat{y}(x_i)||_2^2 + \lambda \sum_{l=1}^{2L} ||W^{(l)}||_F^2
$$

where $L$ is the number of stacked DAs and $W^{(l)}$ denotes weights for the $l$-th layer in the stacked deep network. The sparsity inducing term is not needed for this step because the sparsity was already incorporated in the pre-trained DAs.

### 6.3.5 Image reconstruction

During inference, the test image is first broken up into overlapping $17 \times 17$ patches with stride size of $3 \times 3$. The collection of patches is then passed through LLNet to obtain corresponding denoised, contrast-enhanced patches. The patches are averaged and re-arranged back into its original dimensions. From our experiments, we find that using a patching stride of $2 \times 2$ or even $1 \times 1$ (fully overlapped patches) do not produce significantly superior results. Additionally, increasing the number of DA layers improves the nonlinear modeling capacity of the network. However, a larger model is more computationally expensive to train and we determined that the current network structure is adequate for the present study.

### 6.4 Evaluation metrics and compared methods

In this section we present brief descriptions of other contrast-enhancement methods along with the performance metric used to evaluate the proposed framework’s performance.

#### 6.4.1 Performance metric

Two metrics are used, namely the peak signal-to-noise ratio (PSNR) and the structural similarity index (SSIM).
Figure 6.2: Training the LLNet: Training images are synthetically darkened and added with noise. These images are fed through LLNet where the reconstructed images are compared with the uncorrupted images to compute the error, which is then back-propagated to finetune and optimize the model weights and biases.

6.4.1.1 Peak signal-to-noise ratio (PSNR)

PSNR quantifies the extent of corruption of original image with noise as well as approximating human perception of the image. It has also been established to demonstrate direct relationship with compression-introduced noise [115]. Roughly, the higher the PSNR, the better the denoised image especially with the same compression code. Basically, it is a modification of the mean squared error between the original image and the reconstructed image. Given a noise-free $m \times n$ monochrome image $I$ and its reconstructed version $K$, MSE is expressed as:

$$\text{MSE} = \frac{1}{mn} \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} [I(i,j) - K(i,j)]^2$$  \hspace{1cm} (6.5)

The PSNR, in decibels (dB) is defined as:

$$\text{PSNR} = 10 \cdot \log_{10} \left( \frac{\text{max}(I)^2}{\text{MSE}} \right)$$  \hspace{1cm} (6.6)

Here, $\text{max}(I)$ is the maximum possible pixel value of the two-dimensional image denoted by the matrix $I$. 
6.4.1.2 Structural similarity index (SSIM)

SSIM is a metric for capturing the perceived quality of digital images and videos [87, 159]. It is used to measure the similarity between two images. SSIM quantifies the measurement or prediction of image quality with respect to initial uncompressed or distortion-free image as reference. As PSNR and MSE are known to quantify the absolute error between the result and the reference image, such metrics may not really quantify complete similarity. On the other hand, SSIM explores the change in image structure and being a perception-type model, it incorporates pixel inter-dependencies as well as masking of contrast and pixel intensities. Recall that, from section 3.5.2, the SSIM is expressed as:

\[
\text{SSIM}(x, y) = \frac{(2\mu_x\mu_y + c_1)(2\sigma_{xy} + c_2)}{(\mu_x^2 + \mu_y^2 + c_1)(\sigma_x^2 + \sigma_y^2 + c_2)}
\]  

(6.7)

where \(\mu_x\) is the average of window \(x\), \(\mu_y\) is the average of window \(y\), \(\sigma_x^2\) is the variance of \(x\), \(\sigma_y^2\) is the variance of \(y\), \(\sigma_{xy}\) is the covariance of \(x\) and \(y\), \(c_1 = (k_1L)^2\) and \(c_2 = (k_2L)^2\) are two variables to stabilize the division with weak denominator with \(k_1 = 0.01\) and \(k_2 = 0.03\) by default, and \(L\) is the dynamic range of pixel values.

6.4.2 Compared methods

This subsection describes several low-light image enhancement methods used for comparison. While we acknowledge other recent non-DL methods [39, 31, 158, 153], the lack of publicly available source codes prevented us from performing detailed comparison.

6.4.2.1 Histogram equalization (HE)

The histogram of an image is a graphical representation of the intensity distribution of the image which quantifies the number of pixels for each of the intensity values ranging from 0 to 255 when represented with an 8-bit integer. It is a method that improves the contrast of an image by stretching out the intensity range [143, 26, 103]. It maps the original histogram to another distribution with a wider and more uniform distribution (i.e. flatter) so that the intensity values are spread over the entire range. This method is useful in images with backgrounds and foregrounds that are either both bright or both dark, but may not be suitable for images with
high dynamic range. In particular, the method can lead to better views of bone structure in x-ray images, and to better detail in photographs that are over or under-exposed.

**6.4.2.2 Contrast-limiting adaptive histogram equalization (CLAHE)**

Contrast-limiting adaptive histogram equalization differs from ordinary adaptive histogram equalization in its contrast limiting. In the case of CLAHE, the contrast limiting procedure has to be applied for each neighborhood from which a transformation function is derived [102], as opposed to regular histogram equalization which is carried out in a global manner. CLAHE was developed to prevent the over-amplification of noise that arise in adaptive histogram equalization.

**6.4.2.3 Gamma adjustment (GA)**

The simple form of gamma correction is outlined in Eqn. (6.2). Gamma curves illustrated with $\gamma > 1$ have exactly the opposite effect as those generated with $\gamma < 1$. It is important to note that gamma correction reduces toward the identity curve when $\gamma = 1$. In other words, any image corrected with $\gamma = 1$ results in the exact same image. As discussed earlier in section 6.3.3, the image is generally brightened when $\gamma < 1$ and darkened when $\gamma > 1$.

**6.4.2.4 Histogram equalization with 3D block matching (HE+BM3D)**

BM3D is the current state-of-the-art algorithm for image noise removal presented by [28]. It uses a collaborative form of Wiener filter for high dimensional block of patches by grouping similar 2D blocks into a 3D data array, and then denoising the grouped patches jointly. The denoised patches from the stack are applied back on the original images by a voting mechanism which removes noise from the considered region.

In this work, we decided to first equalize the contrast of the test image, then use BM3D as a denoiser to remove the noise resulting from histogram equalization. Previously, we also attempted to reverse the order, i.e. use BM3D to remove noise from the low-light images first and followed by contrast enhancement. Since BM3D removes noise by applying denoised patches, the blob-shaped patch boundaries are significantly amplified and become extremely pronounced when histogram equalization is applied. This produces non-competitive results which make
comparison unfair. Hence, we ensure that BM3D is performed after histogram equalization when reporting the results.

6.5 Results and discussion

In this section, we evaluate the performance of our framework against the methods outlined above on standard images shown in Fig. 6.3. Test images are darkened with $\gamma = 3$, where noisy versions contain Gaussian noise of $\sigma = 18$ and $\sigma = 25$, which are typical values for image noise under poor illumination and/or high temperature; these parameters correspond to scaled variances of $\sigma_s^2 = 0.005$ and $\sigma_s^2 = 0.010$ respectively if the pixel intensities are in 8-bit integers ($\sigma_s = \sigma/255$ where $\sigma_s \in [0,1]$ and $\sigma \in [0,255]$). These parameters are first fixed in order to study the effectiveness of each method in contrast enhancement and denoising. For a more generalized set of synthetic test images, darkening and noise addition are performed using randomized values of $\gamma \in [1,4]$ and $\sigma \in [0,25]$.

Histogram equalization is performed by using the MATLAB function `histeq`, whereas CLAHE is performed with the function `adapthisteq` with default parameters ($8 \times 8$ image tiles, contrast enhancement limit of 0.01, full range output, 256 bins for building contrast enhancing transformation, uniform histogram distribution, and distribution parameter of 0.4). Gamma adjustment is performed on dark images with $\gamma = 1/3$ unless otherwise stated. For the hybrid ‘HE+BM3D’ method, we first applied histogram equalization to enhance image contrast before using the BM3D code developed by Dabov et al. [28] as a denoiser, where the noise standard deviation input parameter for BM3D is set to $\sigma = 25$ (the highest noise level of the test image). Both LLNet and S-LLNet outputs are reconstructed with overlapping $17 \times 17$ patches of stride size $3 \times 3$. Training was performed on NVIDIA’s TITAN X GPU using Theano’s deep learning framework [10, 16] and took approximately 30 hours. Enhancing an image with dimension of $512 \times 512$ pixels took 0.42 s on GPU.

6.5.1 Algorithm adaptivity

Ideally, an already-bright image should no longer be brightened any further. To test this, different enhancement algorithms are performed on a normal, non-dark and noiseless image.
Table 6.1: PSNR and SSIM of outputs using different enhancement methods. ‘Bird’ means the non-dark and noiseless (i.e. original) image of Bird. ‘Bird-D’ indicates a darkened version of the same image. ‘Bird-D+GN18’ denotes a darkened Bird image with added Gaussian noise of $\sigma = 18$, whereas ‘Bird-D+GN25’ denotes darkened Bird image with added Gaussian noise of $\sigma = 25$. Bolded numbers corresponds to the method with the highest PSNR or SSIM. Asterisk (*) denotes our framework.

<table>
<thead>
<tr>
<th>Test Items</th>
<th>Dark</th>
<th>HE</th>
<th>CLAHE</th>
<th>GA</th>
<th>HE+BM3D</th>
<th>LLNet*</th>
<th>S-LLNet*</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bird</td>
<td>N/A</td>
<td>11.22 / 0.63</td>
<td>21.55 / 0.90</td>
<td>8.93 / 0.66</td>
<td>11.27 / 0.69</td>
<td>17.61 / 0.84</td>
<td>18.35 / 0.85</td>
</tr>
<tr>
<td>Bird-D</td>
<td>12.27 / 0.18</td>
<td>11.28 / 0.62</td>
<td>15.15 / 0.52</td>
<td>29.53 / 0.86</td>
<td>11.35 / 0.71</td>
<td>20.09 / 0.69</td>
<td>15.87 / 0.52</td>
</tr>
<tr>
<td>Bird-D+GN18</td>
<td>12.56 / 0.14</td>
<td>9.25 / 0.09</td>
<td>14.63 / 0.11</td>
<td>14.10 / 0.11</td>
<td>9.98 / 0.13</td>
<td>20.17 / 0.66</td>
<td>18.59 / 0.54</td>
</tr>
<tr>
<td>Bird-D+GN25</td>
<td>12.70 / 0.12</td>
<td>9.04 / 0.08</td>
<td>13.60 / 0.09</td>
<td>13.07 / 0.08</td>
<td>9.72 / 0.11</td>
<td>21.87 / 0.64</td>
<td>22.53 / 0.63</td>
</tr>
<tr>
<td>Girl</td>
<td>N/A</td>
<td>18.24 / 0.80</td>
<td>17.02 / 0.76</td>
<td>11.05 / 0.81</td>
<td>18.23 / 0.69</td>
<td>15.17 / 0.77</td>
<td>14.31 / 0.72</td>
</tr>
<tr>
<td>Girl-D</td>
<td>12.27 / 0.18</td>
<td>11.28 / 0.62</td>
<td>15.15 / 0.52</td>
<td>29.53 / 0.86</td>
<td>11.35 / 0.71</td>
<td>20.09 / 0.69</td>
<td>15.87 / 0.52</td>
</tr>
<tr>
<td>Girl-D+GN18</td>
<td>12.56 / 0.14</td>
<td>9.25 / 0.09</td>
<td>14.63 / 0.11</td>
<td>14.10 / 0.11</td>
<td>9.98 / 0.13</td>
<td>20.17 / 0.66</td>
<td>18.59 / 0.54</td>
</tr>
<tr>
<td>Girl-D+GN25</td>
<td>12.70 / 0.12</td>
<td>9.04 / 0.08</td>
<td>13.60 / 0.09</td>
<td>13.07 / 0.08</td>
<td>9.72 / 0.11</td>
<td>21.87 / 0.64</td>
<td>22.53 / 0.63</td>
</tr>
<tr>
<td>House</td>
<td>N/A</td>
<td>13.86 / 0.70</td>
<td>18.89 / 0.81</td>
<td>10.21 / 0.59</td>
<td>13.24 / 0.64</td>
<td>11.35 / 0.55</td>
<td>10.52 / 0.46</td>
</tr>
<tr>
<td>House-D</td>
<td>12.12 / 0.33</td>
<td>12.03 / 0.65</td>
<td>16.81 / 0.60</td>
<td>28.79 / 0.83</td>
<td>11.92 / 0.54</td>
<td>21.80 / 0.64</td>
<td>18.31 / 0.46</td>
</tr>
<tr>
<td>House-D+GN18</td>
<td>12.19 / 0.29</td>
<td>10.55 / 0.33</td>
<td>15.48 / 0.35</td>
<td>14.44 / 0.34</td>
<td>11.39 / 0.42</td>
<td>21.01 / 0.57</td>
<td>19.31 / 0.47</td>
</tr>
<tr>
<td>House-D+GN25</td>
<td>12.16 / 0.26</td>
<td>10.09 / 0.29</td>
<td>14.08 / 0.29</td>
<td>13.26 / 0.29</td>
<td>10.94 / 0.37</td>
<td>20.68 / 0.54</td>
<td>19.84 / 0.47</td>
</tr>
<tr>
<td>Pepper</td>
<td>N/A</td>
<td>18.61 / 0.90</td>
<td>18.27 / 0.76</td>
<td>10.29 / 0.72</td>
<td>18.61 / 0.84</td>
<td>10.53 / 0.66</td>
<td>10.03 / 0.64</td>
</tr>
<tr>
<td>Pepper-D</td>
<td>10.45 / 0.37</td>
<td>18.45 / 0.85</td>
<td>15.46 / 0.61</td>
<td>32.97 / 0.90</td>
<td>18.45 / 0.80</td>
<td>21.52 / 0.79</td>
<td>19.27 / 0.70</td>
</tr>
<tr>
<td>Pepper-D+GN18</td>
<td>10.45 / 0.19</td>
<td>14.69 / 0.21</td>
<td>14.49 / 0.17</td>
<td>15.74 / 0.22</td>
<td>16.97 / 0.57</td>
<td>22.76 / 0.68</td>
<td>22.07 / 0.64</td>
</tr>
<tr>
<td>Pepper-D+GN25</td>
<td>10.41 / 0.15</td>
<td>13.67 / 0.15</td>
<td>13.31 / 0.13</td>
<td>14.33 / 0.16</td>
<td>15.96 / 0.36</td>
<td>22.94 / 0.61</td>
<td>23.17 / 0.61</td>
</tr>
<tr>
<td>Town</td>
<td>N/A</td>
<td>17.55 / 0.79</td>
<td>16.45 / 0.69</td>
<td>10.92 / 0.76</td>
<td>17.70 / 0.76</td>
<td>16.28 / 0.80</td>
<td>16.03 / 0.78</td>
</tr>
<tr>
<td>Town-D</td>
<td>10.17 / 0.36</td>
<td>17.55 / 0.79</td>
<td>15.00 / 0.65</td>
<td>36.80 / 0.97</td>
<td>17.72 / 0.76</td>
<td>21.42 / 0.75</td>
<td>19.90 / 0.68</td>
</tr>
<tr>
<td>Town-D+GN18</td>
<td>10.19 / 0.18</td>
<td>14.85 / 0.25</td>
<td>13.34 / 0.17</td>
<td>15.53 / 0.24</td>
<td>17.51 / 0.41</td>
<td>19.85 / 0.64</td>
<td>20.52 / 0.59</td>
</tr>
<tr>
<td>Town-D+GN25</td>
<td>10.21 / 0.14</td>
<td>14.22 / 0.20</td>
<td>12.40 / 0.13</td>
<td>14.08 / 0.17</td>
<td>16.62 / 0.32</td>
<td>21.63 / 0.60</td>
<td>22.89 / 0.58</td>
</tr>
<tr>
<td>Average</td>
<td>10.95 / 0.24</td>
<td>14.22 / 0.48</td>
<td>15.26 / 0.43</td>
<td>18.65 / 0.51</td>
<td>15.18 / 0.54</td>
<td>19.66 / 0.67</td>
<td>18.86 / 0.61</td>
</tr>
</tbody>
</table>

Table 6.2: Average PSNR and SSIM over 90 synthetic and 6 natural test images. Synthetic test images are randomly darkened with $\gamma \in [1, 4]$ and Gaussian noise levels of $\sigma \in [0, 25]$. Natural test images are taken under natural low-light conditions. Because gamma darkening is performed randomly for this set of images, we search for the optimal $\gamma$ parameter that results in the highest SSIM ($\gamma = 0.05 : 0.05 : 1$) when applying gamma adjustment. Note that searching for the optimal parameter is infeasible in reality because no reference image is available. The number reported within the parentheses is the number of winning instances among 90 synthetic test images and 6 natural test images. Asterisk (*) denotes our framework.

<table>
<thead>
<tr>
<th>Test Items</th>
<th>Dark</th>
<th>HE</th>
<th>CLAHE</th>
<th>GA</th>
<th>HE+BM3D</th>
<th>LLNet*</th>
<th>S-LLNet*</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average PSNR (dB), synthetic</td>
<td>15.7902</td>
<td>13.7765 (0)</td>
<td>14.3188 (5)</td>
<td>15.2692 (6)</td>
<td>10.1603 (20)</td>
<td>19.8109 (52)</td>
<td>18.2248 (7)</td>
</tr>
<tr>
<td>Average SSIM, synthetic</td>
<td>0.4111</td>
<td>0.3524 (3)</td>
<td>0.3255 (1)</td>
<td>0.4345 (2)</td>
<td>0.5127 (17)</td>
<td>0.6912 (65)</td>
<td>0.6066 (2)</td>
</tr>
<tr>
<td>Average PSNR (dB), natural</td>
<td>8.2117</td>
<td>11.7194 (0)</td>
<td>9.9473 (0)</td>
<td>14.6664 (4)</td>
<td>11.9596 (1)</td>
<td>15.1154 (2)</td>
<td>14.4951 (3)</td>
</tr>
<tr>
<td>Average SSIM, natural</td>
<td>0.1616</td>
<td>0.2947 (0)</td>
<td>0.3611 (0)</td>
<td>0.5338 (0)</td>
<td>0.5437 (2)</td>
<td>0.6152 (3)</td>
<td>0.5467 (1)</td>
</tr>
</tbody>
</table>
Fig. 6.3: Original standard test images used to compute PSNR.

Fig. 6.4A shows the result when running the ‘Town’ image through various algorithms. LLNet outputs a slightly brighter image, but not to the degree that everything appears over-brightened and washed-out like the output of GA if GA is blindly applied with $\gamma = 1/3$. This shows that in the process of learning low-light features, LLNet successfully learns the necessary degree of required brightening that should be applied to the image. However, when evaluating contrast enhancement via visual inspection, histogram equalization methods (i.e. HE, CLAHE, HE+BM3D) provide superior enhancement given the original image. When tested with other images (namely, ‘Bird’, ‘Girl’, ‘House’, ‘Pepper’, etc.) as shown in Table 6.1, HE-based methods generally fared slightly better with higher PSNR and SSIM.

6.5.2 Enhancing artificially darkened images

Fig. 6.4B shows output of various methods when enhancement is applied to a ‘Town’ image darkened with $\gamma = 3$. Here, LLNet achieves the highest PSNR followed by GA, but the the other way round when evaluated with SSIM. The high similarity between the GA-enhanced image with the original is expected because the optimal gamma readjustment parameter essentially reverses the process close to the original intensity levels. In fact, when tested with other images, the highest scores for darkened-only images are achieved by only one of LLNet, S-LLNet or GA where HE, CLAHE, and HE+BM3D fail. Results tabulated in Table 6.1 highlight the advantages and broad applicability of the deep autoencoder approach with LLNet and S-LLNet.
Figure 6.4: Comparison of methods of enhancing ‘Town’ when applied to (A) original already-bright, (B) darkened, (C) darkened and noisy ($\sigma = 18$), and (D) darkened and noisy ($\sigma = 25$) images. Darkening is done with $\gamma = 3$. The numbers with units dB are PSNR, the numbers without are SSIM. Best viewed on screen.

6.5.3 Enhancing darkened images in the presence of synthetic noise

To simulate dark images taken with regular or subpar camera sensors, Gaussian noise is added to the synthetic dark images. Fig. 6.4C and 6.4D presents a gamma-darkened ‘Town’ image corrupted with Gaussian noise of $\sigma = 18$ and $\sigma = 25$, respectively. For these test images, both LLNet and S-LLNet attained superior PSNR and SSIM over other methods, as shown in Table 6.1. Histogram equalization methods fail due to the intensity of noisy pixels being equalized and produced detrimental effects to the output images. Additionally, BM3D is not able to effectively denoise the equalized images with parameter $\sigma = 25$ since the structure of the noise changes during the equalization process.
Figure 6.5: Comparison of methods on randomly darkened/noise-added synthetic test images of (A) Bird, (B) Girl, (C) Pepper, and (D) House. Darkening and noise addition are done using randomized values of $\gamma \in [1, 4]$ and $\sigma \in [0, 25]$. The numbers with units dB are PSNR, the numbers without are SSIM. Best viewed on screen.
Figure 6.6: Comparison of methods of enhancing naturally dark images of (A) chalkboard, (B) computer, (C) objects, (D) chart, (E) cabinet, and (F) writings. Selected regions are enlarged to demonstrate the denoising and local contrast enhancement capabilities of LLNet. HE (including HE+BM3D) results in overamplification of the light from the computer display whereas LLNet was able to avoid this issue.
Instead of using fixed values of $\gamma$ and $\sigma$ for darkening and noise addition, we generated 90 images using randomized values of $\gamma \in [1, 4]$ and $\sigma \in [0, 25]$. Next, the performance of each algorithm is evaluated on these 90 images and the average PSNR and SSIM are computed and tabulated in Table 6.2. Four out of the 90 results are shown in Fig. 6.5. In the table, both the average SSIM and PSNR of the standalone LLNet achieved the best performance compared to other methods, and generally fares better than S-LLNet. It appears that S-LLNet only produces the best enhancement at very dark and high noise levels. However, when the $\gamma$ and $\sigma$ parameters vary at slightly lower levels, then LLNet outperforms S-LLNet. This is because LLNet performs both contrast enhancement and denoising simultaneously, rather than doing the tasks in a stage-wise manner which implicitly assumes independence between the two tasks.

### 6.5.4 Application on natural low-light images

When working with downloaded images, a clean reference image is available for computing PSNR and SSIM. However, reference images may not be available in real life when working with naturally dark images. Since this is a controlled experiment, we circumvented the issue by mounting an ordinary cell-phone (Nexus 4) camera on a tripod to capture pictures in an indoor environment with both lights on and lights off. The picture with lights on are used as the reference images for PSNR and SSIM computations, whereas the picture with lights off becomes the natural low-light test image. Although the bright pictures cannot be considered as the ground truth, it provides a reference point to evaluate the performance of various algorithms. Performance of each enhancement method is shown in Fig. 6.6. While histogram equalization greatly improves the contrast of the image, it corrupts the output with a large noise content. In addition, the method suffers from over-amplification in regions where there is a very high intensity brightness in dark regions, as shown by blooming effect on the computer display in panel 6.6B(vi) and 6.6B(vii). CLAHE is able to improve the contrast without significant blooming of the display, but like HE it tends to amplify noise within the images. LLNet performs significantly well with its capability to suppress noise in most of the images while improving local contrast, as shown in the magnified patches at the bottom of Fig. 6.6.
6.5.5 Training with Gaussian vs. Poisson noise

In certain natural low-light scenarios, the underlying noise profile can be properly modeled by photon shot noise or Poisson noise which is a type of electronic noise. The dominant noise in darker regions of an image from an image sensor is usually caused by statistical quantum fluctuations, that is, the variation in the number of photons sensed at a given exposure level. From a mathematical perspective, while Gaussian noise is typically generated separately and added independently to each individual pixels from the original image, Poisson noise takes the original pixel intensities into account and generates new intensities from a Poisson process. In other words, Gaussian noise is independent of the original intensities in the image but Poisson noise is correlated with the intensity of each pixel. A visual example is provided in Fig. 6.7 to show the differences between Gaussian noise and Poisson noise at different light intensities (i.e. photon count). Since most imaging sensors such as CCD and CMOS suffers from Poisson noise when capturing low-light images, training the model with synthetic images with Poisson noise has potential advantages in enhancing natural low-light images. An experimental comparison between two training schemes—with Poisson vs. with Gaussian noise—is presented in Fig. 6.8.

Figure 6.7: The result of adding Gaussian noise and applying Poisson noise with increasing photon count (normalized from 0 to 1 with a step size of 0.1). Although the noise levels between the two noise types look similar at higher photon count, the first three columns look very different. Best viewed on screen.

In Fig. 6.8, the outputs of LLNet trained with Gaussian noise (LLNet-G) generally appear smoother but suffers some loss in detail due to the retention of shadows. On the other hand, the model trained with Poisson noise (LLNet-P) produces comparatively noisier images but with sharper details. The reason contributing to this disparity lies in the nature of the training set.

To explain why LLNet-G tends to retain the shadows but denoises better than LLNet-P, recall that from Fig. 6.7, darker training patches are less affected by Poisson noise compared to
Figure 6.8: Natural low-light image enhancement results of LLNet trained with Gaussian noise (LLNet-G) and Poisson noise (LLNet-P). ‘Gsn’ and ‘Psn’ are abbreviations for Gaussian and Poisson respectively. Best viewed on screen.

Gaussian noise. Therefore, LLNet-G is able to see more noisy training examples (specifically noisy dark patches) and learn how to denoise them. Furthermore, when a very dark patch is corrupted with Gaussian noise, pixel intensities that become negative are clipped to 0. Hence, this raises the average pixel intensity of a single dark training patch and causes this particular patch appear more gray than black. During patch-wise enhancement, LLNet-G may encounter a gray patch and mistake it for a noisy dark patch. Ultimately, the gray patches are darkened and consequently contribute to dark shadows being retained in the enhanced image. On the contrary, dark patches used to train LLNet-P are least affected by Poisson noise, which in turn reduces the number of noisy examples where LLNet-P learns the denoising function from. The resultant effect is a lower denoising capability of LLNet-P compared to LLNet-G, but with the gained advantage where shadows are also enhanced to bring out relevant details.
Note, with sufficient training data and optimized hyperparameters, a deep autoencoder can learn to approximate almost any nonlinear denoising function. On that account, a union of the two training sets (i.e. with Gaussian and Poisson noise) can be used to train a new LLNet where both noise types are taken into consideration. While there are certainly many other ways to further improve the performance of LLNet (e.g. hyperparameter optimization, ensemble methods, and more rigorous process modeling), we show that the notion of transfer learning can be realized with appropriate training data generation schemes that adequately model a real world process. Thus a model trained with synthetic images can indeed be applied to enhance natural low-light images with competitive performance.

6.5.6 Denoising capability, image sharpness, and patch size

There is a trade-off between denoising capability and the perceived sharpness of the enhanced image. While higher PSNR indicates a higher denoising capability, this metric favors smoother edges. Therefore, images that are less sharp often achieve a higher PSNR. Hence, SSIM is used as a complementary metric to evaluate the gain or loss in perceived structural information. From the experiments, a relationship between denoising capability (PSNR), similarity levels (SSIM) and image sharpness is found to be dependent on the dimensions of the denoised patch relative to the test image. A smaller patch size implies finer-grain enhancement over the test image, whereas a larger patch size implies coarser enhancement. Because natural images may also come in varying heights and widths, the relative patch size—a dimensionless quantity that relates the patch size to the dimensions of the test image, $r$—is defined as:

$$r = \frac{d_p}{d_i} = \frac{\sqrt{w_p^2 + h_p^2}}{\sqrt{w_i^2 + h_i^2}}$$

where quantities $d$, $w$, and $h$ denote the diagonal length, width, and height in pixels, with subscripts $p$ and $i$ referring to the patch and test image respectively. Relative patch size may also be thought as the size of the receptive field on a test image. From the results, it is observed that when the relative patch size decreases, object edges appear sharper at the cost of having more noise. However, there exists an optimal patch size resulting in an enhanced image with the highest PSNR or SSIM (as shown in Fig. 6.9 and Fig. 6.10.). If the optimal patch size is
selected based on PSNR, the resulting image will have the lowest noise levels but is less sharp. If the smallest patch size is selected, then the resulting image has the highest sharpness where more details can be observed but with the expense of having more noise. Choosing the optimal patch size based on SSIM produces a more well-balanced result in terms of denoising capability and image sharpness.

We included a natural test image where the US Air Force (USAF) resolution test chart is shown. The test chart consists of groups of three bars varying in sizes labeled with numbers which conforms to the MIL-STD-150A standard set by the US Air Force in 1951. Originally, this test chart is used to determine the resolving power of optical imaging systems such as microscopes, cameras, and image scanners. For the present study, we used this test chart to visually compare the trade-off denoising capability and image sharpness using different relative patch sizes. The results are shown in Fig. 6.10.

![Relative patch size vs PSNR and SSIM](image)

Figure 6.9: Relative patch size vs PSNR and SSIM. The picture with highest PSNR has the highest denoising capability but least sharp. Picture with lowest $r$ has the least denoising capability but has the highest image sharpness. Picture with the highest SSIM balances between image sharpness and denoising capability.

### 6.5.7 Prior knowledge on input

HE can be easily performed on images without any input parameters. Like HE, CLAHE can also be used without any input parameters where the performance can be further finetuned with various other parameters such as tile sizes, contrast output ranges, etc. Gamma adjustment and BM3D both require prior knowledge of the input parameter (values of $\gamma$ and $\sigma$, respectively), thus it is often necessary to finetune the parameters by trial-and-error to achieve the best results.
Figure 6.10: Evaluation on US Air Force (USAF) resolution test chart. There exist optimal relative patch sizes that result in the highest PSNR or SSIM after image enhancement (using LLNet). Note that the result enhanced with histogram equalization is shown to highlight the loss in detail of the natural dark image (where the main light is turned off) compared to the natural bright image.

The advantage of using deep learning-based approach, specifically using LLNet and S-LLNet, is that after training the model with a large variety of darkened and noisy images with proper choice of hyper-parameters, there is no need for meticulous hand-tuning during testing/practical use. This effectively reduces the burden on the end-user. The model automatically extracts and learns the underlying features from low-light images. Essentially, this study shows that a deep model that has been trained with varying degrees of darkening and noise levels can be used for many real-world problems without detail knowledge of camera and environment.
Feature detectors can be visualized by plotting the weights connecting the input to the hidden units in the first layer. These weights are selected randomly.

Random selection of weights from the first layer (feature detectors) and weights from the output layer (feature generators) for the integrated LLNet model trained with a patch size of $21 \times 21$. Patterns in the output weights are similar to patterns in the first hidden layer weights since tied weights are used.

6.5.8 Features of low-light images

To gain an understanding on what features are learned by the model, weights linking the input to the first layer of the trained model can be visualized by plotting the values of the weight matrix as pixel intensity values (Fig. 6.11). In a regular LLNet where both contrast enhancement and denoising are learned simultaneously, the weights contain blob-like structures with prominent coarse-looking textures. Decoupling the learning process (in the case of S-LLNet) allows us to acquire a better insight. We observe that blob-like structures are learned when the model is trained for the task of contrast enhancement. The shape of the features suggest that contrast enhancement considered localized features into account; if a region is dark, then the model brightens it based on the context in the patch (i.e. whether the edge of an object is
Figure 6.13: Random selection of first layer weights from an integrated LLNet model trained in batches of 1000 and 50, respectively. The superior model (i.e. batch size 50) learns features that appear more distinctive.

Table 6.3: Average PSNR evaluated on the set of 90 test images enhanced with trained model of different hyper-parameters and network architecture. The implemented model is marked by an asterisk (*), whereas the PSNR and SSIM for the best model are presented in bolded typeface.

<table>
<thead>
<tr>
<th>Model Description</th>
<th>Network Architecture (# Hidden Units)</th>
<th>PSNR (dB)</th>
<th>SSIM</th>
</tr>
</thead>
<tbody>
<tr>
<td>*Batch Size 50</td>
<td>2000-1600-1200-1600-2000</td>
<td>19.8109</td>
<td>0.6912</td>
</tr>
<tr>
<td>Batch Size 1000</td>
<td>2000-1600-1200-1600-2000</td>
<td>20.0550</td>
<td>0.6600</td>
</tr>
<tr>
<td>Patch Size 13×13</td>
<td>2000-1600-1200-1600-2000</td>
<td>19.8281</td>
<td>0.6271</td>
</tr>
<tr>
<td>*Patch Size 17×17</td>
<td>2000-1600-1200-1600-2000</td>
<td>19.8109</td>
<td>0.6912</td>
</tr>
<tr>
<td>Patch Size 21×21</td>
<td>2000-1600-1200-1600-2000</td>
<td>19.5877</td>
<td>0.6375</td>
</tr>
<tr>
<td>Patch Size 25×25</td>
<td>2000-1600-1200-1600-2000</td>
<td>19.4637</td>
<td>0.6355</td>
</tr>
<tr>
<td>3-layer SDA</td>
<td>1600-1200-1600</td>
<td>19.8109</td>
<td>0.6912</td>
</tr>
<tr>
<td>*5-layer-SDA</td>
<td>2000-1600-1200-1600-2000</td>
<td>20.2458</td>
<td>0.6845</td>
</tr>
<tr>
<td>7-layer SDA</td>
<td>2400-2000-1600-1200-1600-2000-2400</td>
<td>19.1717</td>
<td>0.6480</td>
</tr>
<tr>
<td>Narrowest</td>
<td>500-400-300-400-500</td>
<td>19.3688</td>
<td>0.6774</td>
</tr>
<tr>
<td>Narrow</td>
<td>1000-800-600-800-1000</td>
<td>19.8056</td>
<td>0.6879</td>
</tr>
<tr>
<td>*Regular</td>
<td>2000-1600-1200-1600-2000</td>
<td>19.8109</td>
<td>0.6912</td>
</tr>
<tr>
<td>Wide</td>
<td>4000-3200-2400-3200-4000</td>
<td>19.7214</td>
<td>0.6730</td>
</tr>
</tbody>
</table>

On the other hand, feature detectors for the denoising task appears noise-like, albeit in a finer-looking texture compared to the on coarser ones from the integrated LLNet. These features shows that the denoising task is mostly performed in an overall manner. Note that while the visualizations presented in [22] show prominent Gabor-like features at different orientations for the denoising task, the Gabor-like features are not apparent in the present study because the training data consists of multiple noise levels rather than a fixed one. The distinction between feature detectors and feature generators is highlighted in Fig. 6.12 and a comparison of superior and inferior weights is shown in Fig. 6.13.
6.5.9 Hyper-parameters, network architecture, and performance

Table 6.3 shows the average PSNR and SSIM values evaluated on the set of 90 synthetic dark images enhanced with the trained model of different hyper-parameters and network architecture. As we are interested in the overall image enhancement performance, we used the implementation with the highest SSIM, as opposed to PSNR, for the reported results. From the results, smaller batch sizes result in noisier gradients during the update and may help in escaping local minima during optimization. Hence, we see that the SSIM increases with a sufficiently small batch size. No clear trend is observed in terms of PSNR with varying batch sizes. A patch size of 13×13 resulted in the highest average PSNR whereas a patch size of 17x17 resulted in the highest SSIM. This result has been discussed in earlier sections and is consistent with the findings on the relationship between the relative patch size and PSNR and SSIM, where the selection of the optimal patch size requires considering the trade-off between image sharpness with denoising power. On the other hand, the number of hidden layers must be chosen such that it adequately captures the nonlinearity in the data (i.e. architecture is not too shallow) while avoiding the vanishing gradient problem which inhibits learning (i.e. architecture is not too deep). The same effect is observed for the width of the architecture.

Note that we explored how varying the hyperparameters one at a time affects the model performance. However, if we use all the optimal hyperparameters discovered independently, such model will not necessarily result in a globally optimal performance. Hence, it may be desirable to explore the hyperparameter space randomly [15] as opposed to doing the search in a sequential manner.

6.5.10 Color implementation

LLNet accepts RGB images comprising the red (R), green (G), and blue (B) channels. In the deep SDA implementation, each channel is flattened into row vectors and concatenated horizontally into a single input vector. For example, a 17 × 17 px RGB image has 289 input units per channel and 867 units total after concatenation. These 867 elements are treated as the visible layer in the deep network and trained identically as the process outlined in section 6.3.1.
In this section, we briefly present the enhancement results from the color LLNet and compare it with some of the popular enhancement methods in Fig. 6.14.

Natural images are randomly collected from the Internet, without any accompanying bright counterparts. Hence, the test images come from a variety of different cameras with different exposure settings and sensor quality. In general, color histogram equalization results in colors that are overly saturated and appear unnatural. In Fig. 6.14(b), the blues from the surroundings are exaggeratedly amplified. While Fig. 6.14(a,c,d,e) look visually satisfactory, the noise content in the images is still prominent. On the other hand, gamma adjustment is able to retain a natural color representation without color saturation. However, it also suffers from amplifying image noise. To remove noise, we applied color BM3D (CBM3D [27]) on the histogram-equalized images, and observe that CBM3D at $\sigma = 25$ is still unable to remove most noise as it is generally suited for salt-and-pepper noise, Gaussian noise, Poisson noise, and not quantization noise (e.g. JPEG quantization). However, it does bring a slight visual improvement as seen in Fig. 6.14(e), where some noise from the heads of the people are reduced. Comparatively, LLNet is able to enjoy all these benefits— retaining a natural color representation (less saturation like HE, and washed-out colors from GA) while reducing noise.

6.6 Conclusions and future works

A variant of the stacked sparse denoising autoencoder was trained to learn the brightening and denoising functions from various synthetic examples as filters which are then applied to enhance naturally low-light and degraded images. Results show that deep learning based approaches are suitable for such tasks for natural low-light images of varying degree of degradation. The proposed LLNet framework compete favorably with currently used image enhancement methods such as histogram equalization, CLAHE, gamma adjustment, and hybrid methods such as applying HE first and subsequently using a state-of-the-art denoiser such as BM3D. While the performance of some of these methods remain competitive in some scenarios, our framework was able to adapt and perform consistently well across a variety of (lighting and noise) situations. This implies that deep autoencoders are effective tools to learn underlying signal characteristics and noise structures from low-light images without hand-crafting. Some envisaged improvements
and future research directions are: (i) Training with quantization artifacts to simulate a more realistic situation; (ii) explore other deep architectures for the purpose of natural low-light image enhancement; (iii) include de-blurring capability explicitly to increase sharpness of image details; (iv) train models that are robust and adaptive to a combination of noise types with extension beyond low-light scenarios such as foggy and dusty scenes; (v) perform subjective evaluation by a group of human users.
(a) Campfire. Photo courtesy of Rick Gershon from MediaStorm (Captured with Canon C300).

(b) Docked ship. Photo courtesy of Msavic6 from AndroidCentral (Captured with Samsung Galaxy S5).

(c) Lady and tree. Photo credits and image acquisition equipment unknown.

(d) Food and cups. Photo courtesy of SCSLab (Captured with Nexus 4).

(e) Concert hall. Photo courtesy of ArsTechnica (Captured with iPhone 4S).

(f) Park. Photo courtesy of CUVI. Image acquisition equipment unknown.

Figure 6.14: Enhancing natural low-light colored image with alternative methods. From left to right: Natural dark, histogram equalization, gamma adjustment, histogram equalization with Color-BM3D (CBM3D), and LLNet. Best viewed on screen, in color (full resolution available when zoomed in).
CHAPTER 7. ROOT-CAUSE ANALYSIS FOR TIME-SERIES ANOMALIES VIA SPATIOTEMPORAL CAUSAL GRAPHICAL MODELING

Modern distributed cyber-physical systems encounter a large variety of anomalies and in many cases, they are vulnerable to catastrophic fault propagation scenarios due to strong connectivity among the sub-systems. In this regard, root-cause analysis becomes highly intractable due to complex fault propagation mechanisms in combination with diverse operating modes. This paper presents a new data-driven framework for root-cause analysis for addressing such issues. The framework is based on a spatiotemporal feature extraction scheme for multivariate time series built on the concept of symbolic dynamics for discovering and representing causal interactions among subsystems of a complex system. We propose sequential state switching ($S^3$) and artificial anomaly association ($A^3$) methods to implement root-cause analysis in an unsupervised and semi-supervised manner respectively. Synthetic data from cases with failed pattern(s) and anomalous node are simulated to validate the proposed approaches, then compared with the performance of vector autoregressive (VAR) model-based root-cause analysis. The results show that: (1) $S^3$ and $A^3$ approaches can obtain high accuracy in root-cause analysis and successfully handle multiple nominal operation modes, and (2) the proposed tool-chain is shown to be scalable while maintaining high accuracy.

7.1 Motivation

With the advent ubiquitous sensing, advanced computation and strong connectivity, modern distributed cyber-physical systems (CPSs) such as power plants, integrated buildings, transportation networks and power-grids have shown tremendous potential of increased efficiency,
robustness and resilience. From the perspective of performance monitoring, anomaly detection and root-cause analysis of such systems, technical challenges arise from a large number of subsystems that are highly interactive and operate in diverse modes.

For the purpose of root-cause analysis for time-series anomalies in complex systems, Granger causality is applied to model the system-wide behavior and capture the variation that can be used to implement root-cause analysis [25]. With multivariate time series data, the case study shows that the causality from and to the fault variable presents differences and can be used to reason the root-cause [154, 93, 70]. For anomaly detection in time series, Qiu et. al. [105] derived neighborhood similarity and coefficient similarity from Granger-Lasso algorithm, to compute anomaly score and ascertain threshold for anomaly detection. A causality analysis index based on dynamic time warping is proposed by Li et. al. [77] to determine the causal direction between pairs of faulty variables in order to overcome the shortcoming of Granger causality in nonstationary time series. Fault related variables can be clustered and root-cause analysis is implemented in each cluster. A dynamic uncertain causality graph is proposed for probabilistic reasoning and applied to fault diagnosis of generator system of nuclear power plant [156]. Also, Bayesian network has been applied in cyber-physical systems to implement anomaly detection and root-cause analysis [63]. The proposed approaches provide efficient tools in discovering causality in complex systems, while an approach in inferencing (interpreting the variation in causality into decisions on failed patterns of fault variable/node) is less investigated.

In this context, we present a semi-supervised tool for root-cause analysis in complex systems based on a data driven framework proposed for system-wide time-series anomaly detection in distributed complex system [79], and using a spatiotemporal feature extraction scheme built on the concept of symbolic dynamics for discovering and representing causal interactions between the subsystems. The proposed tool aims to (i) capture multiple operational modes as nominal in complex CPSs, (ii) only use nominal data and artificially generated fault data to train the model without requiring true labeled anomaly data, and (iii) implement root-cause analysis in a semi-supervised way in a diversity of faults (e.g., one failed pattern, multiple failed patterns, one fault node, and multiple fault nodes). We present two approaches for root-cause analysis, namely the sequential state switching ($S^3$, based on free energy concept of an RBM) and artificial
anomaly association \((A^3\), a multi-label classification framework using deep neural networks, DNN). Synthetic data from cases with failed pattern(s) and faulty node are simulated to validate the proposed approaches, then compared with the vector autoregressive (VAR) model in terms of root-cause analysis performance.

7.2 Background and preliminaries

7.2.1 Unsupervised anomaly detection with spatiotemporal graphical modeling

STPN modeling involves partitioning and discretization followed by learning Markov machines. More details can be found in Section 4.4.

A data-driven framework for system-wide anomaly detection is proposed in [79], noted as the STPN+RBM model. The general idea is to form an unsupervised spatiotemporal graphical modeling approach that can recognize the system-wide patterns in distributed complex systems and detect anomaly only with nominal data. The steps of learning the STPN+RBM model are:

1. Learn APs and RPs (individual node behaviors and pair-wise interaction behaviors) from the multivariate training symbol sequences.

2. Consider short symbol subsequences from the training sequences and evaluate \(\Lambda_{ij}\) \(\forall i, j\) for each short subsequence.

3. For one subsequence, based on a user-defined threshold on \(\Lambda_{ij}\), assign state 0 or 1 for each AP and RP; thus every subsequence leads to a binary vector of length \(L\), where \(L = \#AP + \#RP\).

4. An RBM is used for modeling system-wide behavior with nodes in the visible layer corresponding to APs and RPs.

5. The RBM is trained using binary vectors generated from training subsequences.

6. Online anomaly detection is implemented by computing the probability of occurrence of the current state via trained RBM. The anomaly detection process is shown in Fig. 7.1.
7.3 Methods

Based on previous works on anomaly detection, anomaly manifests itself as a low probability event (high energy). Therefore, the idea for root-cause analysis is to find out the potential pattern(s) that if changed can transition the high energy system state back to a low energy state. The probability of the AP and RP’s existences are discovered by the STPN, and an anomaly will influence the causality of specific patterns (e.g., in STPN, the probability of the pattern might be switched/flipped). Hence, by switching/flipping a pattern, its contribution on the energy states of the system can be identified. Using that principle, the failed patterns can be identified for root-cause analysis. In this regard, two approaches—the sequential state switching ($S^3$) and artificial anomaly association ($A^3$)—are proposed, where both of them are also based on the spatiotemporal graphical modeling framework described in [79].

7.3.1 Sequential state switching ($S^3$)

For an $n$-node graphical model, all the APs and RPs together form a binary vector $v$ of length $L = n^2$ ($L = \#AP + \#RP$, where $\#AP = n$, $\#RP = n \times (n - 1)$). One such binary vector is treated as one training example for the system-wide RBM (with $n^2$ number of visible units) and many such examples are generated from different short sub-sequences extracted from the overall training sequence. Then, the RBM is trained by maximizing the maximum likelihood of the data.

Figure 7.1: Anomaly detection process. Spatiotemporal features are extracted from both nominal and anomalous data, with online STPN applied. Multiple sub-sequences of APs and RPs form input vectors to the RBM. Here, the RBM is only trained with nominal data, and the anomalous data is used as input to compute the free energy. Anomaly is detected by identifying its high energy state, which can be captured using the Kullback-Leibler Distance (KLD) metric.
During training, weights and biases are obtained such that the training data has low energy. During inference then, an anomalous pattern should manifest itself as a low probability (high energy) configuration. The energy function for an RBM is defined as:

\[ E(v, h) = -h^T W v - b^T v - c^T h \]

where \( W \) are the weights of the hidden units, \( b \) and \( c \) are the biases of the visible units and hidden units respectively.

With the weights and biases of RBM, free energy can be computed. Free energy is defined as the energy that a single visible layer pattern would need to have in order to have the same probability as all of the configurations that contain \( v \) [50], which has the following expression:

\[ F(v) = -\sum_i v_i a_i - \sum_j \log(1 + e^{b_j + \sum_i v_i w_{ij}}) \]

The free energy in nominal conditions is noted as \( \tilde{F} \). In cases where there are multiple input vectors with more than one nominal modes, free energy in the nominal states can be averaged or used in conjunction with other metrics. In anomalous conditions, a failed pattern will shift the energy from a lower state to a higher state. Assume that the patterns can be categorized into two sets, \( v^{\text{nom}} \) and \( v^{\text{ano}} \). By flipping the set of anomalous patterns \( v^{\text{ano}} \), a new expression for free energy is obtained:

\[ F^s(v) = -\sum_g v_g a_g - \sum_j \log(1 + e^{b_j + \sum_g v_g w_{gj}}) \]

\[ -\sum_h v_h^* a_h - \sum_j \log(1 + e^{b_j + \sum_h v_h^* w_{hj}}), \{v_g\} \in v^{\text{nom}}, \{v_h^*\} \in v^{*,\text{ano}} \]

Here, \( v^* \) has the opposite state to \( v \) and represents that the probability of the pattern has been significantly changed. In this work, the probabilities of the patterns are binary (i.e. 0 or 1). Hence, we have that \( v^* = 1 - v \). Sequential state switching is formulated by finding a set of patterns \( v^{\text{ano}} \) via:

\[ \min(F^s(v^{\text{ano}}, v^{\text{nom}}) - \tilde{F}) \]

Algorithm 1 is presented to perform root-cause analysis based on the STPN+RBM framework. It should be noted that free energy \( F \) is used in Algorithm 1, and it can used along with
Algorithm 1 Root-cause analysis with sequential state switching ($S^3$) method

1: procedure STPN+RBM modeling ⊳ Algorithm 1 in [79]
2:    Online process of computing likelihoods/probabilities of APs & RPs
3:    Training RBM to achieve low energy state, using binary vectors from nominal subsequences
4: end procedure

5: procedure Anomaly detection ⊳ Algorithm 2 in [79]
6:    Online anomaly detection via the probability of the current state via trained RBM
7: end procedure

8: procedure Root-cause analysis
9:    if Anomaly = 1 then
10:       $F^c \leftarrow F^s(v)$ ⊳ $F^c$ is the current free energy with input vector $v = v^{\text{nom}} \cup v^{\text{ano}}$
11:         $\{V'_p\} \leftarrow \{v : F^s(v) < F^c\}$
12:         while $\{V'_p\} \neq \emptyset \lor \{v : F^s(v^{\text{ano}}, v^{\text{nom}}) < F^c\} = \emptyset$ do
13:             $F^c \leftarrow \min(F^s(v^{\text{ano}}, v^{\text{nom}})), v_i \in \{V'_p\}, v^*_i = 1 - v_i, v^{\text{ano}} = 1 - v^{\text{ano}}$
14:             $\{v^{\text{ano}}\} \leftarrow \{v^{\text{ano}}\} \cup \{v : F^s(v^{\text{ano}}, v^{\text{nom}}) = F^c\}$
15:             $\{v^{\text{nom}}\} \leftarrow \{v\} \setminus \{v^{\text{ano}}\}$
16:             $\{V'_p\} \leftarrow \{V'_p\} \setminus \{v : F^s(v^{\text{ano}}, v^{\text{nom}}) = F^c\}$
17:         end while
18:    end if
19:    $\{\Lambda^{\text{ano}}\} \leftarrow \{v^{\text{ano}}\}$
20:    return $\{\Lambda^{\text{ano}}\}$
21: end procedure
other metrics such as KLD. Using KLD alongside with free energy is particularly useful when the distribution of free energy is obtained with multiple sub-sequences. KLD may be more robust as it takes multiple sub-sequences into account because a persistent anomaly across the subsequences will cause a significant impact on KLD.

7.3.2 Artificial anomaly association ($A^3$)

Using a deep neural network (DNN), we frame the root-cause analysis problem as a multi-class (binary-class in our case) classification [84] problem. The input is presented as an $n^2$-element vector with values of either 0 or 1 which denotes whether a specific pattern is activated. We desire to map the input vector to an output vector of the same length (termed as the indicator label), where the value of each element within the output vector indicates whether a specific pattern is anomalous. For nominal modes, the input vector may be comprised of different combinations of 0’s and 1’s, and the indicator labels will be a vector of all 1’s (where the value 1 denotes no anomaly). However, if a particular element $i$ within the input vector gets flipped, then the indicator label corresponding to the $i$-th position in the output vector will be flipped and switches from 1 (normal) to 0 (abnormal). In this way, we can pinpoint that the $i$-th pattern is anomalous. By setting up the problem in such a manner, a classification sub-problem (i.e. is this pattern normal, or anomalous?) can solved for each elements in the output vector given the input data. One might argue that multi-class classification is unnecessary and adds to higher computational overhead. A single class denoting which node has failed may work just as well without expensive computations. However, cases where multiple patterns fail at once are certainly not uncommon.

The crux of the idea is that anomalies are artificially injected into readily-available nominal data since anomalous data are not always available. A justification is that we wish the model to hierarchically extract important features from the nominal data to sufficiently discriminate between nominal and anomalous patterns. Furthermore, realistic physical systems may have multiple nominal nodes and it will most certainly be arduous to discover the root-cause using conventional methods when anomaly occurs. As deep learning methods are able to learn deep representations of any given input data, it becomes an attractive candidate to solve the same
problem where multiple nominal modes are involved.

Figure 7.2: Framing the problem as an artificial anomaly association (A³) problem. (a) When training the model, the value of an element in both the input vector and its corresponding indicator label is randomly flipped to simulate anomaly. (b) When inferencing, a test input is fed into the DNN and a classification sub-problem is solved to obtain the indicator vector. Values of 0 in the output vector traces back to the exact patterns that are faulty.

DNN Parameters: Training data is generated from 6 modes including both nominal and anomalous data, which is artificially generated from nominal data. The training data is split into 222,300 training samples and 74,100 validation samples. Testing was performed on datasets generated from entirely different modes. The DNN is comprised of 3 layers of 500 hidden units each and trained with ReLU activations. A learning rate of 0.1 and a batch size of 10 is used in the gradient descent algorithm. The training procedure employs the early-stopping algorithm where training stops when validation error ceases to decrease.

7.4 Results and discussions

Synthetic datasets are generated with vector autoregressive (VAR) process to simulate anomaly in pattern(s) and node(s). This section evaluates the performance of S³ and A³ methods using these datasets.

7.4.1 Anomaly in pattern(s)

Dataset: Anomaly in pattern(s) is defined as the change of one or more causal relationships, while defining anomaly, this translates to a changed/switched pattern in the context of STPN. A 5-node system is defined and shown in Fig. 7.3 including six different nominal modes. Cycles
(1 → 2 → 5 → 1, 1 → 2 → 3 → 1, 2 → 3 → 2) and loops (4 → 4) are included in the models. The graphical models are applied to simulate multiple nominal modes. Anomalies are simulated by breaking specific patterns in the graph; 30 cases are formed including 5 cases in one failed pattern, 10 cases in two failed patterns, 10 cases in three failed patterns, and 5 cases in four failed patterns. Multivariate time series data (denoted as dataset1) are generated using VAR process that follows the causality definition in the graphical models.

![Graphical models](image)

Figure 7.3: Graphical models defined to simulate pattern(s) anomaly. Six graphs are defined and treated as nominal operation modes in complex systems. Pattern failure is simulated by breaking specific patterns in the model (not shown).

**Performance Evaluation:** Root-cause identification performances of $S^3$ and $A^3$ methods are evaluated using dataset1. The accuracy metric used here is a pattern-by-pattern comparison (both anomalous and nominal) between the prediction and the ground truth labels. Formally, we define the accuracy metric $\alpha_1$ as:

$$\alpha_1 = \frac{\sum_{j=1}^{n^2} \sum_{i=1}^{m} \chi_1(T_{ij} = P_{ij})}{mn^2}$$

where $T_{ij}$ denotes the ground truth state (nominal/anomalous) of the $j^{th}$ pattern of the $i^{th}$ test sample. Similarly, $P_{ij}$ is the corresponding predicted state using the root-cause analysis approach. $\chi_1(\cdot)$ is the indicator function that returns the value 1 if the expression within the function is true. In the denominator, $m$ is the number of test samples and $n^2$ is the number of patterns.
With the above metric, results of $S^3$ method and $A^3$ method are listed in Table 7.1. High accuracy is obtained for both $S^3$ and $A^3$ method. While training time is much less for $S^3$, the inference time in root-cause analysis for $S^3$ is much more than that of $A^3$, as $S^3$ depends on sequential searching. Note, the classification formulation in $A^3$ aims to achieve the exact set of anomalous nodes. On the other hand, the $S^3$ method is an approximate method that sequentially identifies anomalous patterns and hence, the stopping criteria would be critical. The observation that the performance of $S^3$ is quite comparable to that of $A^3$ suggests a reasonable choice of the stopping criteria.

Table 7.1: Root-cause analysis results in $S^3$ method and $A^3$ method with synthetic data.

<table>
<thead>
<tr>
<th>Approach</th>
<th>Training samples</th>
<th>Testing samples</th>
<th>Accuracy $\alpha_1$ (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S^3$</td>
<td>11,400</td>
<td>57,000</td>
<td>97.04</td>
</tr>
<tr>
<td>$A^3$</td>
<td>296,400</td>
<td>57,000</td>
<td>97.42</td>
</tr>
</tbody>
</table>

It is evident from the above discussion that while the approximate $S^3$ method may overestimate the set of anomalous patterns to some extent, it will be important to not miss any anomalous pattern. In this context, we define a more relaxed metric below to understand the efficacy of the $S^3$ method.

$$\alpha_2 = \sum_{i=1}^{m} \chi_1\left(\{A_T\}_i \subseteq \{A_P\}_i\right)$$

where $\{A_T\}_i$ denotes the ground truth set of anomalous patterns for test sample $i$ and $\{A_P\}_i$ denotes the corresponding predicted set. As expected, we obtain a higher accuracy value of 99.02% for this relaxed metric which shows that the approximate $S^3$ approach is reliable in finding out all of the failed patterns.

7.4.2 Anomaly in node(s)

Dataset: Anomaly in node(s) occurs when one node or multiple nodes fail in the system. For a cyber-physical system, they may be caused by sensor fault or component degradation. The graphical model defined in Fig. 7.3 (a) is applied for generating the nominal data using the VAR process. Anomaly data are simulated by introducing time delay in a specific node. The time delay will break most of the causality to and from this node (except possibly the self loop, i.e., AP of the failed node). To validate this, the causality are recovered with VAR in the cases
of faulty nodes, and the results are shown in Fig. 7.4). The figure also shows that there are
some variations in causality between normal nodes which suggests that causality discovery is
more difficult with existence of cycles and loops. The generated dataset is denoted as dataset2.
For scalability analysis, a 30-node system is further defined with VAR process, and the same
method is applied to simulate the anomaly data, noted as dataset3.

![Figure 7.4: Anomalous conditions with fault node, causality is discovered by VAR model. Compared with the nominal mode in Fig. 7.3 (a), the fault node breaks most of the causality from and to this node. Causality is normalized by the maximal value among all of the patterns, and causality smaller than 0.03 is not shown.]

**Performance Evaluation:** This work is aimed at discovering failed patterns instead of
recovering underlying graph (which is very difficult in graphs with cycles and loops [111, 68]).
The discovered anomalous patterns can then be used for diagnosing the fault node. For instance,
a failed pattern $N_i \rightarrow N_j$ discovered by root-cause analysis can be caused by the fault node
$i$ or $j$. However, if multiple failed patterns are related to the node $i$, then this node can be
debemed anomalous. In this regard, it is important to learn the impact of one pattern on a
detected anomaly compared to another. This can facilitate a ranking of the failed patterns
and enable a robust isolation of an anomalous node. However, the exact $A^3$ does not provide
such information and therefore, we investigate only results obtained with the sequential $S^3$
method. For comparison, we use VAR-based graph recovery method that is widely applied in
economics and other sciences, and efficient in discovering Granger causality [41]. Note, the test
dataset itself is synthetically generated using a VAR model with a specific time delay. Hence,
the causality in such a multivariate time series is supposed to be well captured by VAR-based
method. The details of the VAR-based root-cause analysis strategy is explained below.

Consider the time series $Y(t) = y_{i,t}, i = (1, 2, \cdots , n), \ t \in \mathbb{N}$

$$y_{i,t} = \sum_{k=1}^{p} \sum_{j=1}^{n} A_{i,j}^k y_{j,t-k} + \mu_t, \ j = (1, 2, \cdots , n)$$
where \( p \) is time lag order, \( A_{i,j} \) is the coefficient of the influence on \( i \)th time series caused by \( j \)th time series, and \( \mu_t \) is an error process with the assumption of white noise with zero mean, i.e. \( E(\mu_t) = 0 \), and that the covariance matrix \( E(\mu_t\mu'_t) = \Sigma_\mu \) is time invariant.

With the given time series, a VAR model (i.e., the coefficients \( A_{i,j} \)) can be learnt using standard algorithm \[41\]. The differences in coefficients between the nominal and anomalous models are subsequently used to find out the root causes. The pattern is deemed to have failed when \( \delta A_{i,j} > 0.4 \cdot \max \{ \delta A_{i,j} \} \) where \( \delta A_{i,j} = |A_{i,j}^{ano} - A_{i,j}^{nom}| \).

The results of \( S^3 \) and VAR using dataset3 are shown in Fig. 7.5. In panel (a), all of the changed patterns discovered by \( S^3 \) can be attributed to node 1 (shown by the black boxes in column and row 1). Therefore, node 1 is considered as faulty by the \( S^3 \) method. On the other hand, VAR incorrectly discovers a significant change in the pattern \( N_{12} \rightarrow N_2 \) but not the patterns originating from \( N_1 \). We note this as an error. In general, although \( S^3 \) and VAR can both discover the fault node, VAR produces more false alarms. Similar observations can be made in cases (b) and (c). In (d), 22 failed patterns are discovered by \( S^3 \) which can all be attributed to a fault in node 28, but only six failed patterns are discovered by VAR. In this case,
the discovery of more changed patterns can contribute to a stronger evidence that node 28 is faulty. As it is always beneficial to avoid false positive error, we conclude that $S^3$ method is more preferred for case (d).

In real applications, when more anomalous patterns are discovered incorrectly, more effort will be needed to analyze the failed patterns closely and determine the root-cause node. This will lead to more financial expenditures and time investment in finding the failed node. With this motivation, an error metric $\epsilon$ is defined by computing the ratio of incorrectly discovered anomalous patterns $|\{\Lambda^e\}|$ to all discovered anomalous patterns $|\{\Lambda^{ano}\}|$, i.e., $\epsilon = \frac{|\{\Lambda^e\}|}{|\{\Lambda^{ano}\}|}$. The results using dataset2 and dataset3 are listed in Table 7.2.

<table>
<thead>
<tr>
<th>Approach</th>
<th>Dataset 2 (5 nodes)</th>
<th>Dataset 3 (30 nodes)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$</td>
<td>{\Lambda^{ano}}</td>
</tr>
<tr>
<td>$S^3$</td>
<td>13</td>
<td>2</td>
</tr>
<tr>
<td>VAR</td>
<td>20</td>
<td>4</td>
</tr>
</tbody>
</table>

While it should be noted that both $S^3$ and VAR can discover the fault node correctly in both the datasets, the error ratio for $S^3$ method is much lower than that for VAR (i.e., lower false alarm). As the scale of the system increases, the number of discovered anomalous patterns by $S^3$ becomes more than that of by VAR, while the error ratio becomes significantly less than that of VAR. Therefore, $S^3$ method is both scalable as well as demonstrates better accuracy. Note that for comparisons between $S^3$ and VAR, only one nominal mode is considered in Table 7.2 as VAR is not directly applicable in cases with multiple nominal modes. $S^3$ method can handle multiple nominal modes and it has been validated in Section 7.4.1.

Remark 7.4.1 Note that the root-cause analysis algorithms proposed in this work are fundamentally pattern-based as opposed to being node-based. Motivation for such methods come from real cyber-physical systems, where cyber-attacks may only compromise interactions among sub-systems (i.e., relational patterns (RPs)) without directly affecting sub-systems (i.e., the nodes). In contrast, majority of the existing methods mostly focus on node-based anomalies to the best of our knowledge. For example, in [105], root-cause analysis was performed using Granger Graphical Model with neighborhood similarity (GGM-N) and Granger Graphical Model
with coefficient similarity (GGM-C). An initial investigation shows that for the single node failure cases in dataset3, GGM-N and GGM-C can isolate correct anomalous nodes 25 and 24 out of 30 cases respectively. On the other hand, by a simple rule in determining the anomalous node in dataset3 (finding out the anomalous node that can explain all the failed patterns), $S^3$ can discover correct anomalous nodes in all 30 cases. Future work will involve a detail comparison study by formalizing the anomalous node identification process from pattern-based evidences especially for multiple anomalous nodes.

7.4.3 Discussions

*Ability to handle multiple nominal modes.* The STPN+RBM framework is capable of learning multiple modes as nominal, which corresponds to diverse operation modes in most physical systems (e.g. complex cyber-physical systems). The root-cause analysis approaches proposed in this work enjoy this benefit and are validated with dataset1.

*Accuracy.* The proposed approaches–$S^3$ and $A^3$–demonstrate high accuracy in root-causes analysis with synthetic data. Moreover, the approach shows considerable credibility in finding out all of the failed patterns while avoiding false negatives.

*Scalability.* The approaches are scalable with the size of the system. This is important in complex cyber-physical systems where thousands of variables are often involved.

*Robustness.* Compared with VAR model, the proposed approach can more effectively pinpoint the fault node with less incorrectly discovered patterns. $S^3$ method can also find out the maximal set of failed patterns that is beneficial to avoid false positive errors.

7.5 Conclusions

Based on spatiotemporal graphical modelling, this work presents two approaches—the sequential state switching ($S^3$) and artificial anomaly association ($A^3$)–for root-cause analysis in complex cyber-physical systems. With synthetic data, the proposed approaches are validated and showed high accuracy in finding failed patterns and diagnose for the fault node. Compared to the VAR model, the proposed $S^3$ approach can more effectively pinpoint the fault node with less incorrectly discovered patterns. It is also adaptive by design and can be applied to systems
with multiple nominal operation modes. The further work will pursue: (i) inference approach in node failure including single node and multiple nodes, (ii) detection and root-cause analysis of simultaneous multiple faults in distributed complex systems.
CHAPTER 8. SUMMARY AND CONCLUSION

From earlier chapters, we have seen that the integration of machine learning methods—especially deep learning—with domain knowledge holds enormous potential in solving problems dealing with complex dynamical systems, human-machine interaction systems, computer vision, and cyber-physical systems. Specifically, deep neural networks are capable of modeling complicated nonlinearities in the data and serve as a stackable, more powerful method for dimension reduction—similar to PCA—as demonstrated in chapter 7 for root-cause analysis, and chapter 4 for the early detection of combustion instabilities. Restricted Boltzmann machines are also used as part of the framework for identifying the source of anomalies in chapter 7. Additionally, the ability of convolutional neural networks to exploit local correlations in 2D images is tremendously helpful in applications where high-level decisions need to be made based on images, video frames, or any visual representations of data, described in chapter 3 for the design of microfluidic devices, chapter 4 for health management of combustion engines, and chapter 5 on estimating the value of information for decision and control. Finally, stacked denoising autoencoders show promising results in performing patch-based natural low-light image denoising and contrast enhancement as shown in chapter 6. In contrast to other machine learning methods, deep learning has a significant potential in providing scalable and effective solutions to address data analytics and engineering problems involving large volumes of high-dimensional data. Undoubtedly, deep learning will remain as an invaluable tool over the next few years.
Bibliography


denoising autoencoders: Learning useful representations in a deep network with a local


document modeling. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1315–1324. ACM.


