Hockey Pose Estimation and Action Recognition using Convolutional Neural Networks to Ice Hockey

by

Helmut Neher

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Science
in
System Design Engineering

Waterloo, Ontario, Canada, 2018

This thesis consists of material all of which I authored or co-authored: see Statement of Contributions included in this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Statement of Contributions

The following three papers are used in this thesis. They are described below:

H. Neher, M. Fani, D. A. Clausi, A. Wong, and J. Zelek. Pose estimation of player in hockey videos using convolutional neural networks. In *2017 Ottawa Hockey Analytics Conference (OTTHAC)*, Ottawa, Canada, 2017. Best Student Paper.

This paper is incorporated in Chapter 4.

Contributor	Statement of Contribution
H. Neher (Candidate)	Conceptual Design 70%
	Data collection and analysis 50%
	Writing and editing 50%
M. Fani	Conceptual Design 10%
	Data collection and analysis 50%
	Writing and editing 38%
D. A. Clausi	Conceptual Design 10%
	Writing and editing 5%
A. Wong	Conceptual Design 10%
	Writing and editing 5%
J. Zelek	Writing and editing 2%

H. Neher, K. Vats, A. Wong, and D. A. Clausi. A hyper stacked hourglass deep convolutional neural network architecture for joint player and stick pose estimation in hockey. In *15th Conference on Computer and Robot Vision (CRV)*. Toronto, Canada, 2018.

This paper is incorporated in Chapter 5.

Contributor	Statement of Contribution
H. Neher (Candidate)	Conceptual Design 60%
	Data collection and analysis 50%
	Writing and editing 65%
K. Vats	Conceptual Design 30%
	Data collection and analysis 50%
	Writing and editing 25%
D. A. Clausi	Conceptual Design 5%
	Writing and editing 5%
A. Wong	Conceptual Design 5%
	Writing and editing 5%

M. Fani, H. Neher, D. A. Clausi, A. Wong, and J. Zelek. Hockey action recognition via integrated stacked hourglass network. In *2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 85-93, Honolulu, USA, July 2017.

This paper is incorporated in Chapter 6.

Contributor	Statement of Contribution
H. Neher (Candidate)	Conceptual Design 45%
	Data collection and analysis 50%
	Writing and editing 44%
M. Fani	Conceptual Design 45%
	Data collection and analysis 50%
	Writing and editing 44%
D. A. Clausi	Conceptual Design 5%
	Writing and editing 5%
A. Wong	Conceptual Design 5%
	Writing and editing 5%
J. Zelek	Writing and editing 2%

Abstract

Human pose estimation and action recognition in ice hockey are one of the biggest challenges in computer vision-driven sports analytics, with a variety of difficulties such as bulky hockey wear, color similarity between ice rink and player jersey and the presence of additional sports equipment used by the players such as hockey sticks. As such, deep neural network architectures typically used for sports including baseball, soccer, and track and field perform poorly when applied to hockey. This research involves the design and implementation of deep neural networks for both pose estimation and action recognition can effectively evaluate the pose and the actions of a hockey player.

First, a pre-trained convolutional neural network, known as the stacked hourglass network, is used to determine a hockey player′s body placement in video frames, also known as pose estimation. The proposed method provides a tool to analyze the pose of a hockey player via broadcast video which aids in the eventual assessment of a hockey player′s speed, shot accuracy, or other metrics. The algorithm demonstrated to be successful since it identifies on average 81.56% of the joints of a hockey player on a set of test images.

Furthermore, inspired by the idea that modeling the pose of a hockey stick can improve hockey player pose estimation, a novel deep learning computer vision architecture known as the HyperStackNet has been designed and implemented for joint player and stick pose estimation. In addition to improving player pose estimation, the HyperStackNet architecture enables improved transfer learning from pre-trained stacked hourglass networks trained on a different domain. Experimental results demonstrate that when the HyperStackNet is trained to detect 18 different joint positions on a hockey player (including the hockey stick), the accuracy is 98.8% on the test dataset, thus demonstrating its efficacy for handling complex joint player and stick pose estimation from video.

Extending from pose recognition, this research involves the development of an algorithm for accurate recognition of actions for hockey. To perform this action recognition, a convolutional neural network estimates actions through unifying latent pose and action recognition. The action recognition hourglass network, or ARHN, is designed to interpret player actions in ice hockey video using estimated pose. ARHN has three components. The first component is the latent pose estimator, the second transforms latent features to a common frame of reference, and the third performs action recognition. Since no benchmark dataset for pose estimation or action recognition is available for hockey players, we first had to generate such an annotated dataset. Experimental results show action recognition accuracy of 65% for four types of actions in hockey. When similar poses are merged to three and two classes, the accuracy rate increases to 71% and 78%, proving the potential of the methodology for automated action recognition in hockey.

Acknowledgements

I would like to first thank God for giving me strength in my research. Secondly, I wish to thank my supervisors Prof. Alex Wong and Prof. David Clausi for their support, encouragement, and random talks about hockey, machine learning or tangents of recent technologies. Thank you for being wonderful mentors and for the guidance you have provided to a fledgling researcher.

I would also like to thank the people that comprise the Vision and Image Processing Research Group. Your collaborative efforts, love of computer vision, and regular, vibrant conversations encouraged me to seek novel research. In particular, I would like to thank my collaborators Kanav Vats, Mehrnaz Fani and Prof. John Zelek for their contributions in our research.

Finally, I wish to thank my family, my friends, and my committee members.

This work is partly funded by the Natural Sciences and Engineering Research Council of Canada (NSERC).

Dedication

This is dedicated to the one I love, Tamina.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Computer vision, especially human pose estimation and action recognition are exciting fields of research due to the potential applications and the challenges arising in this field. Applications include security to identify people's intent based on gait and actions, health and safety to identify poor technique or assess practices in the work place and sports to analyzing player performance. Human pose estimation is defined by estimating the particular body configuration of a human; to identify the location of limbs when a human performs a pose (standing, sitting, jumping, etc.). Human action recognition is the estimation of identifying a person's actions, albeit walking, jumping, sitting, running, etc. Challenges from pose estimation and action recognition include foreground being similar to the background, irregular movements, occlusion through bulky clothing or equipment and scene. The plethora of applications and the difficult challenges in these fields makes it attractive for researchers to study and contribute to these fields. One application that exudes excitement and a challenge is ice-hockey.

Hockey is a sport where the study of pose estimation and action recognition can benefit players, coaches and analysts alike because this field of research provides a scientific way to evaluate team and player performances. Current methods for hockey analytics, such as statistics, are typically derived manually by watching live or recorded hockey games and are restricted to shot-based or goal-orientated statistics thus limiting the evaluation of player performance such as player speed and technique. These statistics and models mentioned do not assess the abilities and capabilities of a player, which, do not explain how a player can improve; the statistics only portray the effects of the player with respect to a goal. The need for new methods to evaluate player performance in addition to explaining the capabilities of a hockey player is paramount.

Computer vision provides an alternative view. The use of pose estimation for hockey players provides a tool used within the hockey analytics field to later assess player abilities and performance by assessing player statistics such as speed and technique. Practitioners want more accurate information centered on the actual player (i.e., speed and technique) while also using less time consuming methods. Hockey player pose estimation and hockey action recognition are valuable pieces of information that potentially can help coaches in assessing player performance.

Pose estimation provides valuable information since it can be continuously derived from game video as opposed to goal or shot statistics that only occur periodically during the game for only the individual with the puck. Also, pose information can be derived from all players viewed in the video as opposed to the limited gathering of statistics based only on the player shooting. Pose estimation helps in determining a player's actions and evaluating player performance; the ability to understand the pose of a player leads to the overall analysis of that player's performance. Only a small amount of research in pose estimation in ice hockey has been published [38].

In addition, action recognition in computer vision also contributes to understanding player performance. By understanding the actions of each individual player throughout the entire game, coaches and analysts can use this information to evaluate player effectiveness on a team, or even use this information to determine the strength of a team. In addition, understanding how a player acts during many scenarios of the game, such as offense, defense, puck possession, and scrum will help evaluate the level of performance of that player. Although only a limited amount of action recognition or even computer vision research has been done in the field of hockey, action recognition can be applied to hockey analytics to analyze characteristics of hockey players and teams.

1.1 Motivation

The motivation of this thesis is to solve challenging problems inherent in pose estimation and action recognition. Challenges include: bulky equipment, foreground/background similarity, occlusion and motion blur. Typical deep learning models cannot accurately identify human pose and action recognition from hockey datasets using hockey players because of the bulkiness of hockey player equipment that deforms players' body-shape and occlude joints and limbs, causing inaccuracies. Another challenge is that the color of the player's equipment is often similar to the background causing occlusions of joints and limbs; one team's hockey player uniform is white, which is similar in color to the ice and boards background which are also white. Also high speed of movement from skating or shooting

leads to motion blurring in the image that also needs to be compensated. In addition, most research of pose estimation and action recognition utilize 3D pose information, however, due to the nature of hockey, that information is not available, thus posing a tougher problem as only relative 2D joint information is used. All of these challenges cause the problem of pose estimation and action recognition in hockey challenging and are the motivation for this research.

1.2 Statement of Thesis

The goal of this thesis is to tackle the inherent challenges of pose estimation and action recognition which are commonly seen in hockey datasets by studying and developing novel deep learning architectures found in computer vision. Deep learning architectures are applied to automatically identify the pose and actions of a hockey player from game video in order to eventually assess the capabilities of that player. To accomplish this goal, three tasks are evaluated: pose estimation without the use of hand-held object (e.g., hockey stick), pose estimation with a stick, and action recognition. A convolutional neural network (CNN), a computer vision algorithm that is able to learn patterns based on game video is utilized for these goals. The algorithm is designed to estimate the "pose" information of the player, namely the joint (e.g., wrist, shoulder, pelvis, knees, elbows, neck) as well as the hockey stick locations and associated limb positions. After the pose is established, then the hockey action recognition is tested and evaluated.

1.3 Thesis Contributions

The thesis dissertation presents a study to prove the efficacy of pose estimation in various cases and action recognition in the challenging field of hockey. The primary contributions of the dissertation include the following:

- Introduces pose estimation in hockey via the stacked hourglass architecture [40] using broadcast images;
- Introduces novel idea of modelling pose of a hockey stick using transfer learning and latent heat maps;
- Introduces how the use of pose estimation in hockey is extended to action recognition thus implementing the action recognition hockey network (ARHN);

- Provides experimental results in pose estimation for hockey player pose estimation;
- Provides experimental results in modeling pose of the hockey stick for improved hockey player and stick pose estimation; and
- Provides experimental results in the extension of action recognition using pose estimation in hockey using ARHN.

The secondary contributions of the dissertation include:

- Creation of the HARPE dataset [22]; and
- Present novel architectures which include latent heatmaps as feature extractors.

The aforementioned contributions present an important role in proving the efficacy of pose estimation and action recognition in hockey.

1.4 Outline of Thesis

To address the aforementioned thesis contributions, the thesis will begin by providing background information in computer vision techniques in hockey, pose estimation and action Section in Chapter 2. A system overview of the hockey pipeline will follow in Section 3. Chapters 4, 5, and 6 will discuss pose estimation, pose estimation with added joints locations for a hand-held object and action recognition using pose estimation features. Finally, Chapter 7 will describes the conclusions of the thesis.

Chapter 2

Background

The background composes of sections describing the advancements in computer vision applied to hockey (Section 2.1), pose estimation (Section 2.2) and action recognition (Section 2.3). Within the pose estimation section, discussion on traditional computer vision methods for pose estimation in Subsection 2.2.1 and deep learning methods Subsection 2.2.2 will be discussed. Finally the action recognition section is composed of two subsections: current pose-based action recognition techniques used in sports (Subsection 2.3.1) and computer vision research applied to ice hockey (Subsection 2.3.2).

2.1 Computer Vision in Hockey

Computer vision has been applied to hockey for a limited number of tasks such as, player tracking [4] [44] [43], rectification of noisy broadcast videos [25], puck possession event classification [56], dataset development for spectator categorization/people counting [15] and a few results in action recognition [36, 35, 37]. Within various hockey applications, some methods were completed using traditional computer vision methods, while others were completed using deep learning methods.

Cai *et al.* [4] implements player tracking using a particle filter framework and a linear optimization algorithm to track multiple players. Tora *et al.* [56] implements puck possession event classification using a pre-trained convolutional neural network (CNN) to extract features, which is fed to a long short-term memory neural network (LSTM) for event classification. Okuma *et al.* [44] combines particle filter with Adaboost and also uses a self learning approach in [43]. Lu *et al.* [36, 35, 37] in his three papers, uses histogram

of oriented gradients (HOG) descriptors along with hidden Markov model and particle filtering for action recognition.

To conclude, the only extensive research in hockey has been done in player tracking and action recognition. Most of the techniques used in previously mentioned results use HOG and scale-invariant feature transformer (SIFT) features and probabilistic graphical models without deep neural networks (DNN). In fact, this thesis explores the idea of deep neural networks in pose estimation specifically in hockey.

2.2 Pose Estimation

2.2.1 Traditional Computer Vision Methods

Generally, traditional computer vision methods for pose estimation employs deformable parts models, appearance models, conditional random fields, mixture models and linear/non-linear classifiers. In addition, research involved involved in computer vision used histogram of orientated gradients (HOG) and various methods of support vector machines for pose estimation. This section will describe some important methods found in traditional computer vision.

The early beginnings of pose estimation began employing parts based models or pixel-based approaches in a variety of forms. Ramanan [52] applies an edge-based deformable model (using a CRF), to obtain soft estimates of body part positions by defining pixel labels into region types (background, torso, left lower arm, etc.) by learning low-level segmentation cues to build part-specific region models, but also computing no segmentation. Other methods use similar deformable part models and modify the algorithm through employment of grabcuts, a foreground extractor and Gaussian mixture models [24], and SVM and HOG descriptors used as detection [24, 23, 67].

Other models incorporate poselets which are a part or body part of one's pose that are tightly clustered in both appearance and configuration space. In this work, Bourdev *et al.* [2] uses 3D human pose annotations as references for determining the pose. Also, each poselet has examples used for training an SVM classifier used over a tested image to determine the pose of a human.

Some models sought to improve the shortcomings of Felzenszwalb *et al.* [23] and Bourdev *et al.* [2] by improving on the existing algorithm. Johnson *et al.* [30] uses a pictorial structure model (PSM) which models a person as a connected collection of 10 parts (head, torso, upper and lower body limbs). This method employs a mixture of linear SVMs to

capture part appearances represented by HOG descriptors. This paper improves the handling of self-occlusions by clustering the training images in pose space to form a mixture of trees that were not captured in [23]. In addition, to improve [2], a flexible mixture of parts model that captures contextual co-occurrence relations between parts rather than using articulated limb parts was created [67]. In the aforementioned research, each method implements a mixture of deformable parts and uses HOG descriptors as well as a structure SVM to detect pose.

One unique method proposes a multimodal, decomposable graph model for human pose estimation in monocular images that captures a variety of pose modes [53]. It captures pose modes by using two variables, one for the left side and one for the right side of the body. The authors employ a standard linear pairwise conditional random field.

In addition, pose estimation started with improving existing models by tweaking some modular systems, then it continued by combining known parts based models in addition to pixel-based models [49, 31]. Pischulin *et al.* [49] combines flexible spatial models, from [67], and image-conditioned spatial models and incorporates a basic human body model tree-structure in addition to incoporating a pictoral structure model. In addition, both a part based approach for layout type problems such as a PSM, and an image labelling technique involving optimizing a random field graph defined on the image for pixel-wise evaluation were combined in research using HOG descriptors [31].

2.2.2 Deep Learning

The movement in research that shifted research in pose estimation from traditional methods of computer vision was first highlighted through 'DeepPose' by Toshev *et al.* [57] by regressing x,y coordinates of joints using a cascade regression of deep neural networks. This research leads to further research using deep learning using using graphical models, iterative predictions and depth motion cues began.

Typical methods in deep learning use of convolutional neural networks and graphical models [7, 21, 51, 55]. Variations occur based on unary score generation and pairwise comparison of adjacent joints. Iterative predictions for pose estimation include Carreira *et al.* [6] which utilizes Iterative Error Feedback as a method where each pass through the network further refines predictions such that the input are predictions thus requiring multi-stage training, and share weights for each iteration. Another method is to incorporate multi-stage pose machines using convolutional neural networks for feature extraction [61].

Depth and motion cues such as optical flow are used in conjunction with deep learning methods to solve the pose estimation problem [28, 54, 47]. In addition, other methods

such as simultaneous annotations of people are also explored [51, 8]. In addition, work such as performing human part segmentation using fully convolutional networks are also incorporated to improve human pose estimation [45, 34].

Although each neural network method aforementioned contributes to the pose estimation problem, in recent years, state-of-the-art methods solely use heatmaps for pose estimation. The heat map method of using convolutional neural networks was first introduced by Tompson *et al.* [55] that incorporated deep learning methods, such as convolutional neural networks to solve a regression problem of joint locations using Gaussian probability. The network regresses on x,y coordinates by assuming that at each coordinate pair there exists a 2D Gaussian with a small variance and mean centered at that joint location. This regression problem can be visualized using heatmaps, where the probability of the existence of a joint location would be indicated on the region as red, while, the probability that there is no joint located would be indicated on that region by the color blue. Although Tompson *et al.* [55] incorporates in this paper a graphical model, current research with heatmaps utilize deep learning structures to improve performance.

Heat maps in pose estimation is involved in many deep learning architectures. Most commonly, heat maps are used in many flavours of convolutional neural networks [14, 40, 42, 64]. One popular pose estimation architecture which uses heatmaps is the stacked hourglass architecture by Newell *et al.* [40]. The stacked hourglass network incorporates a convolutional network with a multi-context attention mechanism for pose estimation through the use of residual layers. The approach uses convolutional modules in a bottom-up, top-down configuration resembling hourglasses which are then stacked. One variant of the stacked hourglass network attempts to further improve the stacked hourglass by applying a conditional random field (CRF) to fine tune the performance [14]. In addition, Chu *et al.* designed an hourglass residual unit as apposed to maintaining the residual layer in the original stacked hourglass network to leverage heatmaps which are then used as a latent feature and as the output [14]. To further improve the stacked hourglass network, another method incorporates inception-residual networks in replacement of convolutional layers for a more robust feature representation to form what the authors call a fractal network (combination of hourglass networks, residual and inception layers) [42]. Finally, Pyramid Residual Module method is used to enhance invariance [64] while also employing the stacked hourglass network by Newell *et al.* [40].

Another implementation of the heatmaps is using Generative Adversarial Networks (GANs) [9, 13], part-affinity fields [5] and graph modeling [26]. Employs a generative adversarial network using the same stacked hourglass architecture by Newell *et al.* [40] as both the descriminator and generator [13]. Chou *et al.* [9] uses an adversarial approach method that utilizes priors to ensure the network is more structure-aware through the

use of a pose generator, a pose descriminator and a confidence discriminator used as the structure. The output of the network are heatmaps. Cao *et al.* [5] utilizes part affinity fields to detect multi-person 2D pose estimation in addition to confidence maps. Cao *et al.* employs a variant of heat maps called confidence maps which employs a similar mathematical function to a Gaussian distribution where a mean and standard deviation are the inputs. One advantage of this method is that it can detect multiple people in the same scene. Employs a multi-person estimator through the use of a spatial-temporal edge detector with unary costs thus sparsifying the body-part relationship graph to detect and associate body joints of the same person even in a clutter by leveraging temporal information for crowded scenes [26].

Some state-of-the-art research implement the stacked hourglass network by Newel *et al.* in their architecture [13, 14, 42, 64]. One reason is that the hourglass network was able to beat the state-of-the-art in the MPII dataset [1] of that time, and was also able to maintain spatio information by using skip connections, thus opening more research for pose estimation.

2.3 Action Recognition

2.3.1 Pose-based Action Recognition

Many works in action recognition use dense trajectory features including HOG, HOF, and MBH [12, 29, 41, 50, 60] in addition to pose estimation. Pishchulin *et al.* [50] explore combinations of dense trajectories and pose estimation noting that combinations may improve accuracy of action recognition given that the pose estimates are unable to accurately label the pose of a person. Jhuang *et al.* [60] compare dense trajectory, a low/mid-level method, separately against pose estimation, a high-level method, determining that methods incorporating pose features outperform low/mid level feature methods.

One method to incorporate dense trajectories and pose estimation is using AND-OR graph models [29, 41]. One implementation incorporates motion, geometry of joints (pose) and appearance [29]. The model uses HOF/HOG for motion appearance as a part node. The pose node has a projected 3D view and then it is placed into 'different' view nodes (difference viewpoints); this approach is tested using 2D video input and is to help evaluate actions in various viewpoints. Another model incorporates poselets in addition to HOG/HOF within the And-Or graph model [41].

Similarily to incorporating poselets, Desai *et al.* [16] presents an approach based on combining three compositional models (i.e., poselets, visual phrases, and pictorial structure

models) for modeling human pose and interacting objects. Phraselets are introduced and employed in a Flexible Mixture of Parts (FMP) framework to capture relations between parts and a separate compositional model per action class is defined. Output of the model are action labels, articulated human pose, object pose, and occlusion flags. Phraselets, like most of the methods in action recognition, are designed for recognizing coarse actions that are quite different in nature (such as horse riding verses taking photo), not for fine action recognition (e.g., discriminating between two different movements of a hockey player).

Iqbal and Gall [58] introduce a method for repeatedly alternating between pose estimation and action recognition. They adopt standard pictorial structure model (PS model) for human pose estimation and condition it on action types to do efficient inference. Starting with uniform prior on all action classes, the pose in each frame is predicted, and by using the estimated poses, the probabilities of the actions are estimated.

Recently, deep structures have dominated most of previous descriptors and models for pose estimation and are giving promising results in action recognition.

Chéron *et al.* [12] developed a pose based CNN that incorporates a descriptor for action recognition tasks. Pose estimation is performed using a method given by Cherian *et al.* [11], and is utilized for determining four different regions or body parts in images. Next, optical flow and raw image pixels over patches of body parts are given to separate CNNs to generate motion and appearance descriptors for each frame. Descriptors per frame, and their consecutive differences in successive frames, are aggregated by max and min pooling over time and normalized to generate static and dynamic video descriptors, which are concatenated to form P-CNN descriptor. Besides P-CNN descriptors, three different Improved Dense Trajectory features (i.e., HOG, HOF,and MBH) with Fisher vector coding are also computed. Action recognition is performed using a linear SVM over P-CNN descriptors and IDT features. This method as explained does not use the pose estimation directly as a feature but rather employs it for determining the region of interest for patch selection from images, while pose information, if determined precisely, is intrinsically a strong clue for action recognition.

Similar research in action recognition in sports uses pose estimation as a latent variable in a unified action recognition in still images [65]. Like Yang *et al.* [65], the authors seek to unify pose estimation and action recognition to improve action recognition performance.

This literature review shows pose can be used as a strong feature for action recognition and employing power architectures such as deep networks will increase the accuracy of pose-based action recognition. Therefore, in this work a pose-based deep network incorporating latent pose estimation is implemented for action recognition.

Zhou *et al.* [70] incorporates privilege information which is a shape-based representation data used solely in training using a support vector machine (SVM) with radial basis function (RBF) kernels. This method creates a dense human pose as apposed to highlighting specific joints. This method is trained on still images and action recognition is performed.

Fan *et al.* [20] is similar to the previous research as they use what they term as 'high-level pose features', which, instead of using shapes they incorporate other higher level pose information by relation of pairs and triplets of joints as well as trajectories. In addition, they model the energy change, the relative position of joints and the spatial and temporal space by calculating the inner product of triplet of joints. Fan *et al.* uses videos and applied a k-means codebook and an SVM with RBF kernel to train the classifiers for action recognition.

Du *et al.* [17] uses an end-to-end model that captures both pose estimation features and action recognition for videos. The recurrent pose-attention network (RPAN) implements a recurrent neural network (RNN) to apply to temporal features. The network extracts human-part features, in which a pose loss from the attention heat maps are refined and in which that information is applied to an LSTM which predicts the actions. This end-to-end model simultaneous predicts pose and utilizes pose for action recognition.

Papadopoulos *et al.* [46] implements a dense trajectories approach to extract spatio-temporal characteristics using a bag-of-words implementation of action recognition for video to estimate 2D poses.

2.3.2 Computer Vision and Action Recognition in Hockey

Within the sport of hockey, computer vision research has been limited to tracking [4, 33, 36, 43, 44] , rectification of broadcast hockey video [25], crowd analysis [15] providing a hockey crowd dataset, and very few results in action recognition [35, 36, 37].

In the three papers by Lu *et al.* [35, 36, 37], HOG descriptors are used with various training methods such as support vector machines, prior information is extracted from videos and sequences of images are implemented as input for action recognition. The mentioned papers, however, do not describe methods for extracting higher level features such as pose. The activities evaluated from the aforementioned papers are based on actions of skating such as skating left, skate right, skate in, skate out, skate left 45, and skate right 45 rather than other actions of hockey that focus on the whole body.

This summary represents the limited extent of the published research in the field of

computer vision applied to ice hockey. In this work, a significant state-of-the-art contribution by developing a methodology to automatically determine actions of a hockey player based on latent pose estimation derived from video frames is presented.

Chapter 3

System Overview - Hockey Analysis Pipeline

To understand the how pose estimation and action recognition can be implemented within ice-hockey, the general framework is discussed in Section 3.1. In addition a discussion on the type of data required to be processed within the hockey pipeline is in Section 3.2. Following the hockey data section, the pose estimation, in Section 3.3, and action recognition, in Section 3.4, pipeline in hockey is explained. The output of the data pipeline, Section 3.5, is also discussed.

3.1 Hockey System Overview

Hockey analytics in computer vision uses a general data pipeline system shown in Figure 3.1 to extract data such as actions and pose. The general pipeline begins by inputting images, and then using that information to infer pose. In some cases, raw data can be used to infer actions of players or pose from a pose estimator can be used to infer actions. Both pose estimation and action recognition are then used for analytics to be used in the hockey industry.

The pose and action portions of the general pipeline are systems themselves. The overall goal of this system is to create statistical information or visualization tools to effectively assess a player's ability, skill level and potential for many application which include coaching, health assessments and scouting.

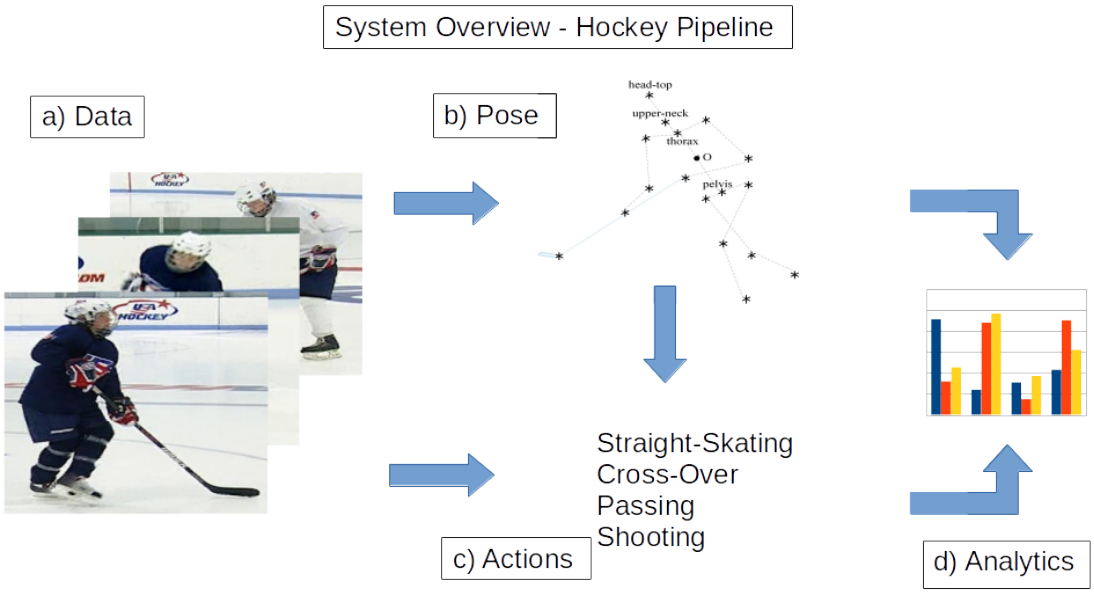


Figure 3.1: Image Data (a) is first extracted from raw images or video sequences found from video sources including broadcast and training videos, then fed to either (b) a pose estimation system or (c) action recognition system. In addition, the pose estimation system may also be used in the action recognition system. The result from both the pose estimator and the action recognizer produce (d), the analytics of the player, line or team for assessment.

3.2 Hockey Data

Hockey data is an integral part of the hockey analytics pipeline as it is the information that will be passed from each portion of the pipeline. Hockey data, in a computer vision context, is comprised of images or sequence of images with some meta data if any. With hockey as an application of computer vision, that same guideline applies. Images used in the data range from single player or goalie to teams. In addition, various types of images such as broadcast images or still action shots of players can also be used for data. Images in a dataset for hockey typically range from images of a single hockey player or goalie, to multiple players of the same team or opposing teams. In addition, the data can be composed of images comprised of action shots of live games or of skills and drills player practice. Also, images can be extracted from broadcast images with wide and steep angle of view to images taken from other sources of video cameras such as a phone or hand-held camera with a smaller field of view and at an angle close to parallel of the ice. Examples of each aforementioned portion of images are shown. Some metadata of an image could include added information to aid the algorithm which could include xy coordinates of the center of each player, a player jersey number and name associated with the player in question, the player's team affiliation and line affiliation.

The data is the input to the next portions of the hockey pipeline, which include a pose estimator and an action recognizer. The data is processed specifically for these inputs and may vary depending on the algorithm used for the estimator and recognizer. In addition, some information such as xy tracking coordinates may be used in addition to help the estimator or recognizer.

3.3 Hockey Pose Estimation

Pose estimation is the act of estimating the body configuration of a hockey player by estimating the location from a 2D image of joints. Generally, within the pose estimator, either an image or hand-crafted features are used as an input in the estimator and xy coordinates of the amount of joints estimated are the output. With respect to the most common and current method of pose estimation, the following figure demonstrates in the basic pipeline. The pipeline consists of an input, which is generally a raw image, followed by a computer algorithm or machine learning neural network of choice then the output are heatmaps of joint locations, which the xy coordinates are then extracted. The pose pipeline is shown in Figure 3.2.

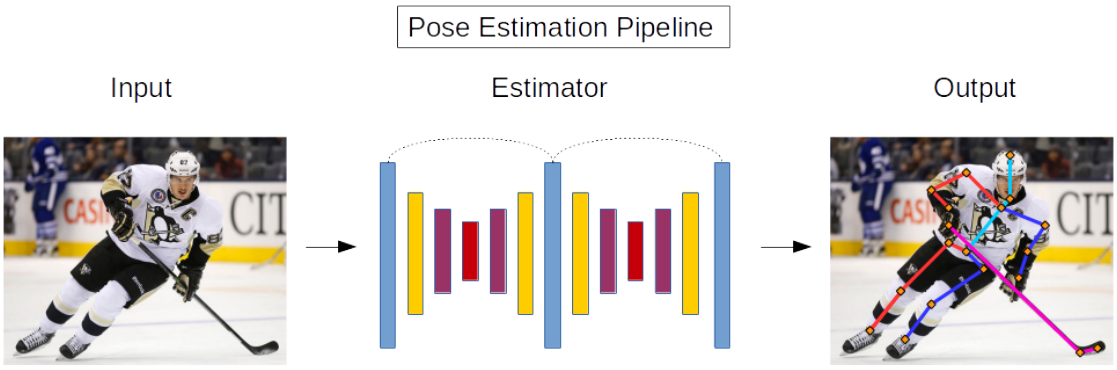


Figure 3.2: The pose estimation pipeline consisting of an image input extracted from video of hockey players, the estimator composed of classical computer vision algorithms or a deep neural network, and the estimated pose as the output.

3.4 Hockey Action Recognition

The action recognition system is very simple in its structure and can be shown in Figure 3.3. The action recognizer begins by an input which is normally of an image or feature extracted by hand or via a computer algorithm or machine learning neural network, followed by the action recognizer algorithm then outputting a class label of a specific action.

3.5 Hockey Output

The output of the general hockey pipeline is the analytics resulted from using the hockey data and processed via computer algorithms. The output are quantifiable metrics for player's, lines, teams or other important aspects of a hockey game. The metrics are to improve the qualitative visual inspections most analysts, scouts and other hockey personnel. Metrics composed from actions and pose can be the following:

- player's speed
- player's technique
- player interaction

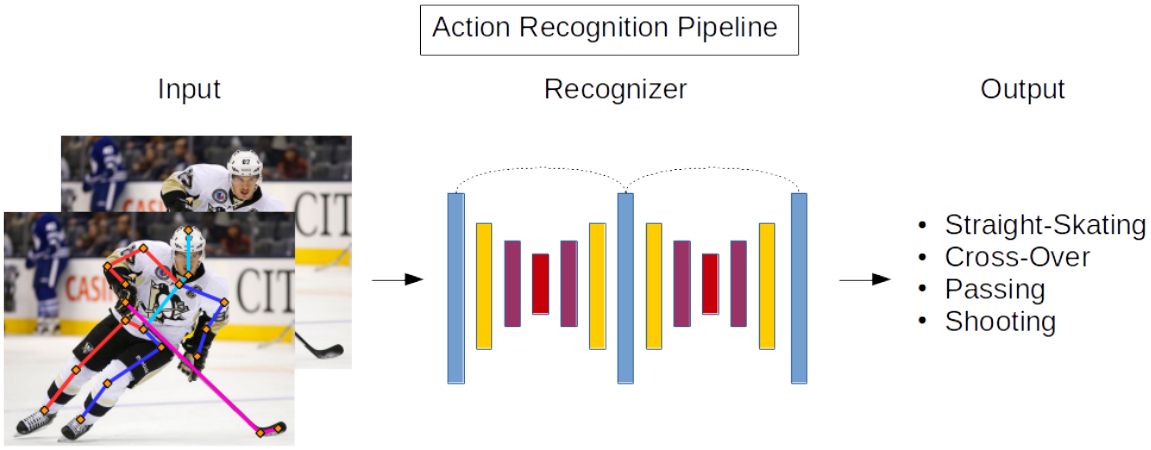


Figure 3.3: General action recognition pipeline consists of an input that can be a combination of raw image data, estimated pose or other features fed into an action recognizer composed of classical computer vision algorithms or a deep neural network that then outputs a classification of an action.

- situation awareness
- actions of a player during game play
- passing accuracy
- shooting accuracy
- player's shot speed

In this thesis, this section is assumed to be completed elsewhere, and the first two portions of the hockey analytics pipeline are expanded and conversed upon.

Chapter 4

Pose Estimation

Extracting the pose of a player is the gateway to provide analysts and coaches with the ability to automatically assess player performance via performance statistics based on speed and technique to evaluate the capability of a player. The goal of this section is to automatically determine the pose of a hockey player from game video in order to eventually assess the capabilities of that player. To accomplish this goal, a convolutional neural network (CNN) is utilized as described in this section which learns patterns based on game video and still images. The algorithm is designed to estimate the "pose" information of the player, namely the joint (e.g., wrist, shoulder, pelvis, knees, elbows neck) locations and associated limb positions. Architectural overview (Section 4.1) will then be discussed followed by the evaluation and testing and results (Section 4.2).

4.1 Architectural Overview

As seen in Figure 4.1, to estimate the pose of a hockey player, six processes occur. An image is first captured from a broadcast hockey video, then the location of the player's body center is automatically determined through a computer vision algorithm that tracks the body center. The image and the coordinate of the body center are then rescaled to an optimal resolution for the CNN (i.e., 720 x 1280 pixels) and cropped to isolate the desired player. Next, the scaled and cropped image, along with body center point is delivered into the hourglass network for pose estimation. The output of the network are 16 joint coordinates that indicated the predicted place for a joint. Afterwards, the body limbs are obtained by connecting related joints together. Finally, an output is visually seen with color-coded limbs in the image.

The output of joint coordinates for players in each frame are valuable pieces of information that can later be used to evaluate player performance such as a player's position, movement, speed, and reaction to a play for a whole broadcasted hockey game. At this aim, the vehicle to determine the joint placement is a convolutional neural network for finding places of joints of a hockey player in each frame of a hockey video or in a single image of a hockey player.

The employed convolutional neural network, shown in Figure 4.2a, is called the stacked hourglass network [40], and is composed of several layers of artificial neurons that are interconnected to each other. The weights of the interconnections, are previously learned from a dataset is the MPII Human Pose Dataset, which is composed of 40,000 images, annotated for 16 body-joints [1]. This network is trained to learn the place of joints for images of people doing daily activities. The output of the network is a set of heatmaps.

Heatmaps are first introduced in Toshev *et al.* [57] and are a regression strategy in neural networks used to isolate a specific point in an image. The network's input is an image, while the regression output is a set of x and y coordinates of joint locations. Figure 4.2b shows an example of a heatmap. Heatmaps are named that way to refer to the visualization process. The final point assumes a Gaussian distribution with a specified standard deviation. This regression can be visualized by using heatmaps where the blue refers to no probability of joint locations, while the red indicates a high probability of a joint location. The higher the probability, the more red it becomes. Thus, it is similar to how infrared or thermal imaging cameras work by indicating red as 'hot' and blue as 'cold'. Regression is completed by the prescribed neural network.

Each heatmap is an image which gives the predicted probability of a joint's presence at each and every image pixel [1]. In Fig. 4.2 an example of a heatmap is provided that predicts the location of left shoulder in the image. Here, pixels with blue color are less probable to be assigned to the left elbow, while for the pixels that are near the elbow location, the heatmap color gradually grows from blue to yellow and finally to red. The 16 body joints that are identified by this network are listed in Table 4.1.

4.2 Experimental Testing and Results

In this part, different experiments are conducted to show the accuracy and effectiveness of the proposed methodology by evaluating the efficacy of the stacked hourglass network in determining the estimated pose of a hockey player. The results are categorized in three groups: results on still images, results on videos, and finally numerical results.

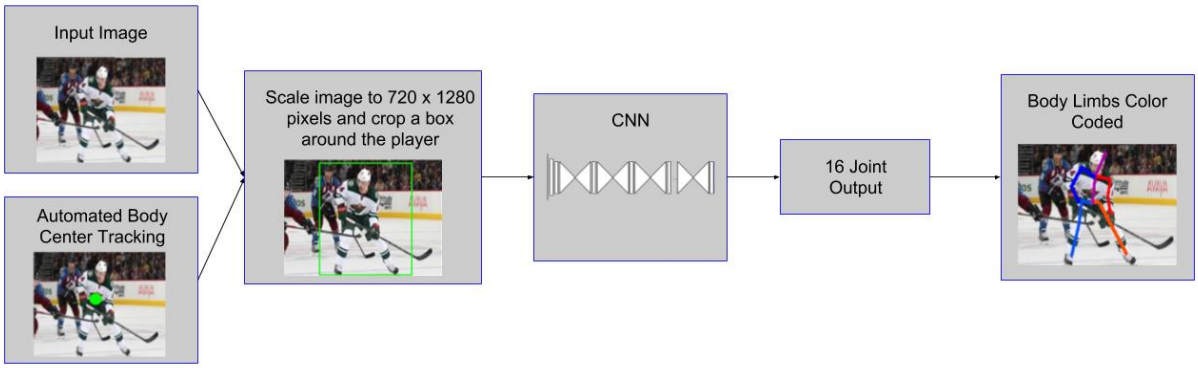
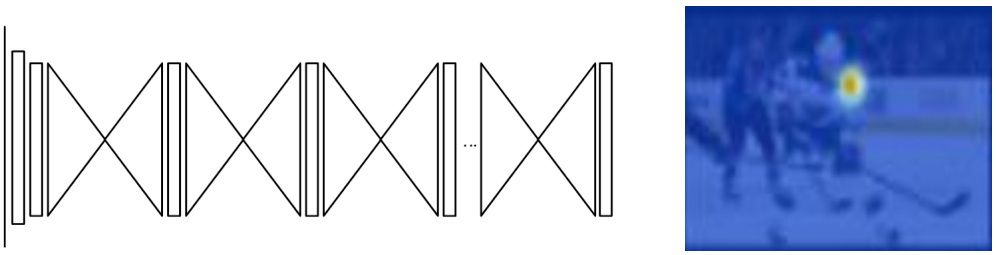


Figure 4.1: The proposed framework for hockey pose estimation. The input to the CNN is a set of trained properly scaled images and a known tracked body center for the player. The CNN output is the 16 joint coordinates color coded for ease of viewing.



(a) (b)

Figure 4.2: a) The stacked hourglass network which is the employed CNN for this research. b) Instance of a heatmap that shows the predicted joint placements for the left shoulder of a hockey player.

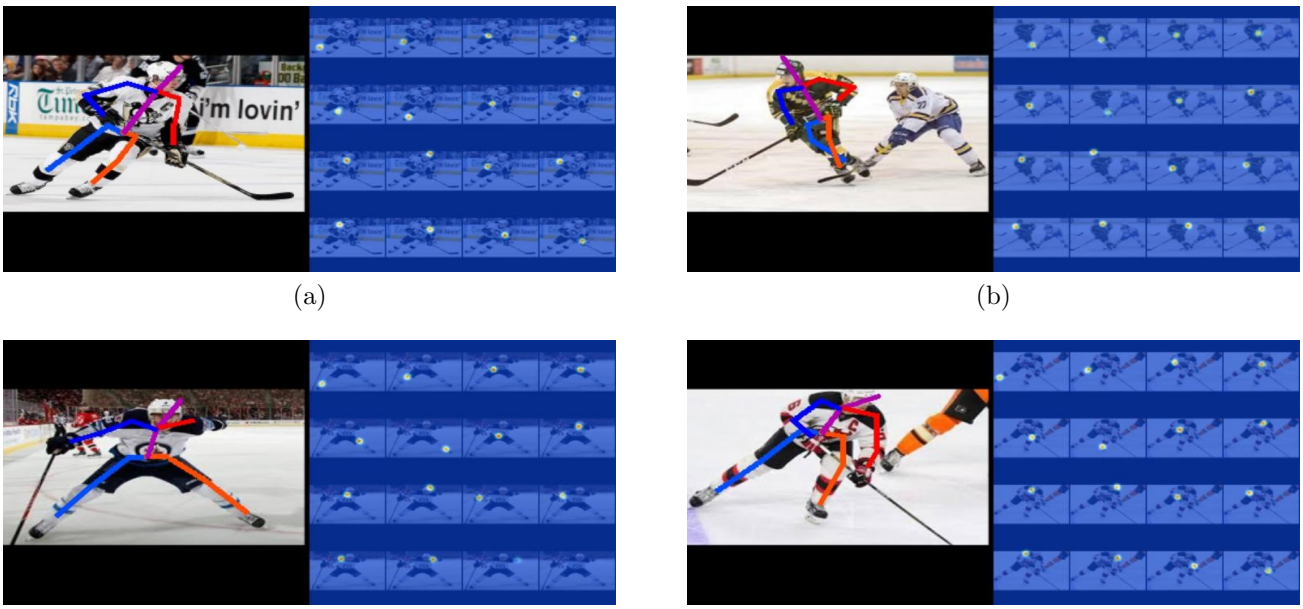
Table 4.1: Joint locations used in pose estimation

Joint	Location	Joint	Location
Right Ankle	1	Upper Neck	9
Right Knee	2	Head Top	10
Right Hip	3	Right Wrist	11
Left Hip	4	Right Elbow	12
Left Knee	5	Right Shoulder	13
Left Ankle	6	Left Shoulder	14
Pelvis	7	Left Elbow	15
Thorax	8	Left Wrist	16

4.2.1 Test Results on Still Images

In the first part of experiments, the framework of Figure 4.1 is applied on 20 still images of hockey players from a corpus of images with various body poses. Four example output images along with their corresponding heatmaps are given in images of Figure 4.3. From these images, joints have been found successfully in each. The 16 heatmaps that are provided for each image, show the predicted probability for location of each joint, with the same order that is given in Table 4.1. In the heatmaps, the color blue corresponds to low probability, while growing to yellow and then red shows the increment of probability for a joint to be situated in a specific position in the image.

From visual inspection, the predictions are accurate; each image demonstrates by the color-coded limbs the general pose of the hockey player. In fact, hockey sticks and bodies of other players do not affect or skew the accuracy of the results; the limbs and hockey sticks of other players are not recognized as the limbs of the player being evaluated. Some joints, however, are not accurately identified such as the ankle joints of the player in question; the equipment of a hockey player may obscure the joints from the CNN. Although, some of the joints may not be accurately located, the CNN successfully identifies the general pose of the hockey player being evaluated without interference from other players within the still image.



(a) (b)

(c) (d)

Figure 4.3: Visual demonstration of hockey pose estimation along with corresponding heatmaps for four still images of hockey players from a corpus of images collected.

Table 4.2: Accuracy of joint detection for 20 hockey player test images

Joint Name	Head	Shoulder	Elbow	Wrist	Hip	Knee	Ankle	Total
Accuracy	90%	87.5%	70%	65%	80%	82.5%	90%	81.56%

4.2.2 Test Results on Video

In the second experiment conducted, Figure 4.1 is applied on a video of 114 frames, (with frame size of 640 x 480, and frame rate of 25 f/sec). Some of the output frames are illustrated in images of Figure 4.4, for visual assessment. In Figure 4.4, most of the body joints are detected with high accuracy even in the frames where the right arm is hidden behind the player's body (i.e., frames 13, 16, 19, and 22). There are still some inaccurate joint locations, but in general, the player's pose is accurately assessed. Therefore, the test video demonstrates the capability of the proposed procedure in pose estimation.

4.2.3 Numerical Results

The percentages of correctly identified joints for 20 still images of hockey players are reported in Table 4.2. According to Table 4.2, the highest accuracies of detection achieved are for head and ankle joints, which are both 90%, while, wrists and elbows are detected with lowest precisions, i.e., 65% and 70% respectively. Generally, the equipment a player wears, namely skates and gloves, hides the joints from the algorithm that adversely affects the precision of the estimated pose. However, as given in the last column of Table II, the average of correct detections for all body-joints is 81.56%. Since the overall percentage indicates that most of the limbs are detected per image, pose estimation is a valuable tool to extract data from video and still images.

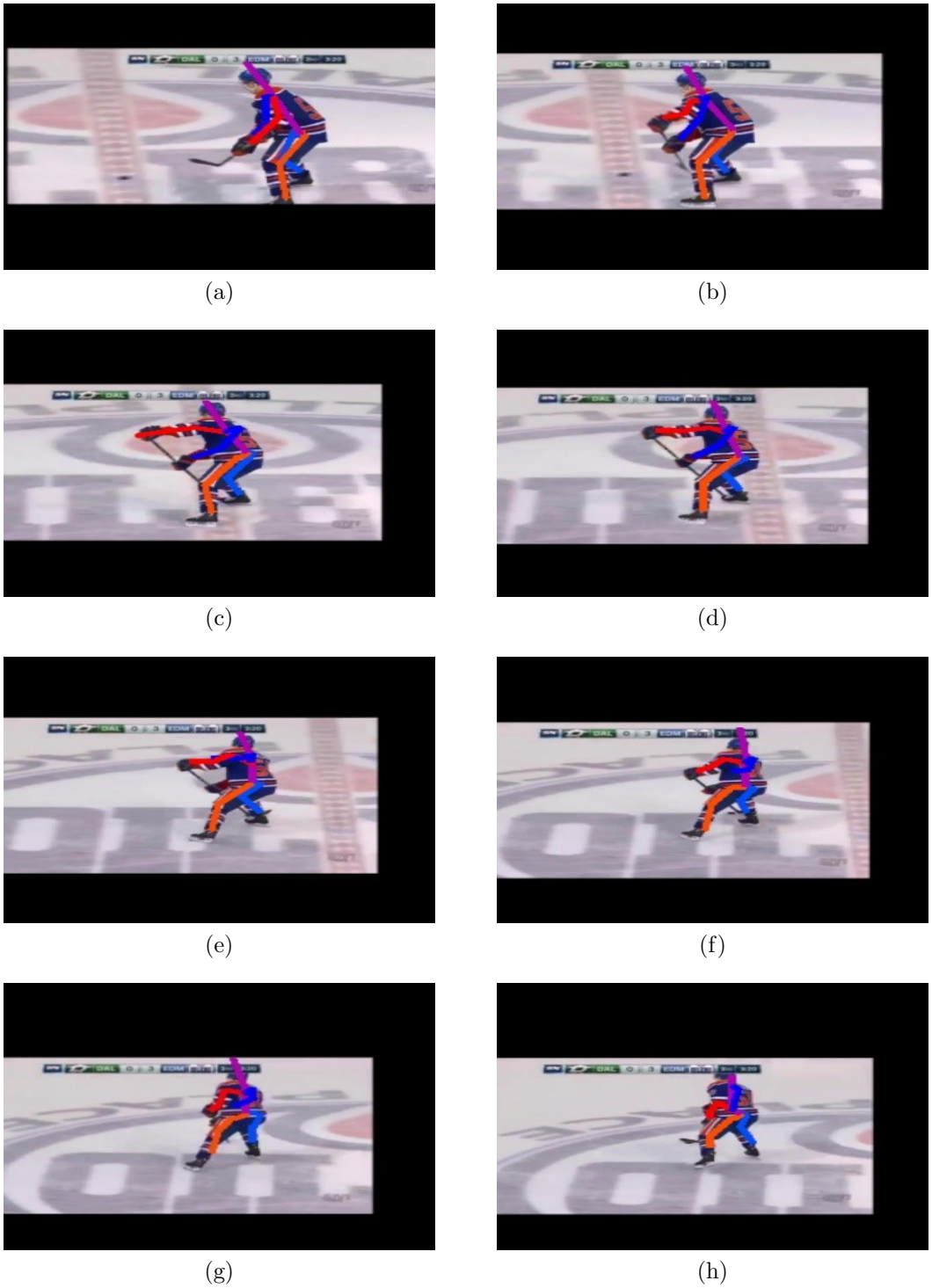


Figure 4.4: Visual demonstration of hockey pose for eight frames of a 114 frame hockey video. Images (a - h) correspond to frames 1, 4, 7, 10, 13, 16, 19, and 22.

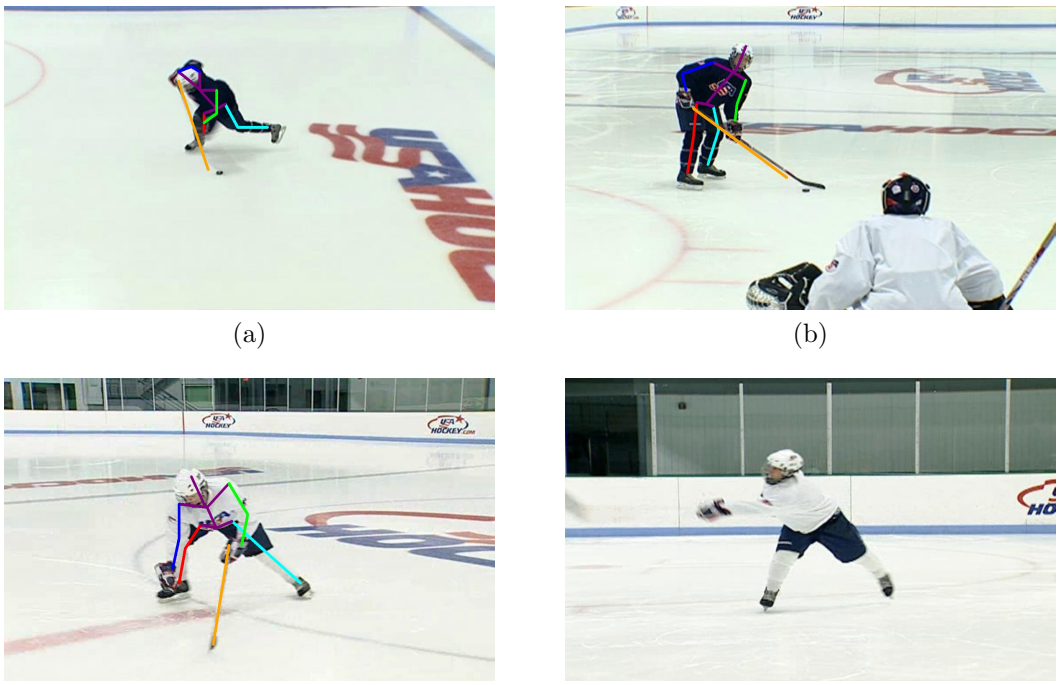
Chapter 5

Pose Estimation with Added Joints

From the previous section, pose estimation plays an important part in sports by providing data and analysis to aid in determining a player's actions and evaluating player performance in ice hockey [38]. Many challenges are associated with ice hockey in pose estimation such as bulky equipment, limited number of datasets, motion blur, foreground/background similarities. All of these challenges affect the performance of pose estimation. To improve the quality of pose estimation mentioned in the previous section, a new approach is needed to continue to tackle ice hockey.

That approach is to improve pose estimation in ice hockey by utilizing a unique feature found in the hands of hockey players - the hockey stick. This handheld object is an extension to the human body used to control, steal, pass, or shoot the puck. By modeling additional joints representing the pose of a hockey stick, that information extracted can be used to infer other joints from the hockey player; by adding a hockey stick, a player's arms are usually constrained to holding that object. Estimating the pose of hockey stick locations not only can improve the pose estimation of hockey players, but it can also help assess a player's actions and performance. By knowing the location of a hockey stick, technique of a player can be evaluated more finely such as hand-eye-coordination, shooting, and passing. Hockey sticks, however, pose certain challenges such as motion blur in video because they are much harder to identify due to their thinness (see Figure 5.1). Inspired by the idea that the position of a hockey stick is useful for both improving player pose estimation and assessing player performance, a novel HyperStackNet network architecture [39] is proposed for joint player and stick pose estimation in this paper.

Yao *et al.* [68] claim that handheld objects (such as bats, balls and rackets) and body can help infer the location of each other for pose and action recognition. Although research



(a) (b)

(c) (d)

Figure 5.1: Results for different player poses (a) - (c). Note the excessive size of equipment posing difficulty for pose estimation. The final image (d) is a case of excessive motion blur. Also note that the jersey color matches with the background, causing difficulty in pose estimation.

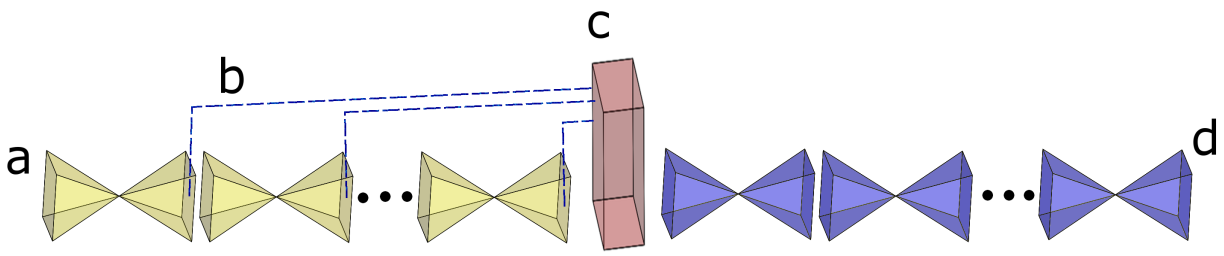


Figure 5.2: The network architecture of HyperStackNet. In the HyperStackNet network architecture, an initial stacked hourglass network (a) is used to model a subset of keypoints. The output tensor of each hourglass network module (b) in this initial stacked hourglass network combines into a latent vector (c), which acts as an intermediate connection to a second stacked hourglass network module that jointly models the previous subset of keypoints plus additional keypoints. In this particular implementation of HyperStackNet, the initial stacked hourglass network module consists of eight hourglass modules modeling 16 joints, resulting in a latent pose vector that combines eight tensors of size 3x16x64x64. The second stacked hourglass network module also consists of eight hourglass modules modeling not only the 16 joints on a player but also an additional two joints associated with the hockey stick, resulting in a final output of 18 different joints.

has been done to detect/track the position of a handheld racket itself [19, 18], to our best knowledge, there is currently no published research that uses an external handheld object for human pose estimation, which would include objects used for playing sports such as sticks and rackets.

Therefore, two additional joints for the hockey stick for pose estimation is considered. This differs from Yao *et al.* [68] in the way that our work focuses on the handheld object (stick) as a dependent part/joint of the human body, so that the mutual information can be exploited.

The stacked hourglass architecture [40] is extended to incorporate two additional joints (total of 18). The reason for using this architecture is that it does not require a separate graphical model as in Chen *et al.* [7], or separate pipeline for resolutions [55], or separate pipeline to associate body parts as in Pishchulin *et al.* [51]. It also finds other applications in [3, 62] and extensions in [14, 63]. In Section 5.1, the details of a unique transfer learning that extends the capacity/domain of the network is introduced. This same concept of extending pose to include a hockey stick can be extended to other racket/bat based sports such as field hockey, cricket and baseball.

The main contribution of this section is the introduction of the HyperStackNet for

Figure 5.3: Some results of HyperStackNet on the test dataset with various player poses.

improved joint hockey player and stick pose estimation. Using the HARPE dataset [22], the HyperStackNet architecture extends significantly upon the stacked hourglass architecture [40] by enabling the latent modeling of pose from pre-trained stacked hourglass networks while at the same time augmenting additional stick pose information to provide improved player pose estimation through the use of novel transfer learning. See Figure 5.2 for the overall architecture.

Section 5.1 will be by providing the architectural overview and Section 5.2 will be the experimental testing and results.

5.1 Architectural Overview

5.1.1 Overview

The HyperStackNet architecture is comprised of three components in the network and will be discussed as follows: 1) the original stacked hourglass network proposed by Newell *et al.* [40] with the input being an image and center of body (Secction 5.2.2), 2) the final output of the first half of the architecture being the latent pose vector which concatenates each hourglass module's output (Section 5.2.3) and 3) a modified stacked hourglass network which takes the latent pose vector of 16 joints as input and trains two more joints locations, outputting 18 joint locations (Section 5.2.4). These three components compose the HyperStackNet as shown in Figure 5.3. The method of transfer learning (Section 5.2.5) will also be discussed in addition to the dataset used (Section 5.2.6) and training details (Section 5.2.7).

5.1.2 Original Stacked Hourglass Network

The first half of the HyperStackNet architecture is similar in structure to the stacked hourglass network described by Newell *et al.* [40]. The network is comprised of eight hourglass modules in sequential order where the input for the first module is an image with an x and y coordinate indicating the center of mass. The output of each hourglass module is a heatmap of the original 16 joints. The final heatmap is formed by applying two consecutive rounds of 1x1 convolutions. To connect modules together, the output of each module, being heatmap predictions of each joint, are then used as inputs for the next hourglass module. The intuition is to train the architecture to model a subset of keypoints at each hourglass module to be used later to improve pose estimation over a larger set of joint locations.

Each hourglass module is comprised of convolutional, max-pooling, and up-sampling layers in addition to skip connections in a bottom-up and then top-down configuration. The bottom-up portion of the hourglass utilizes the convolutional layers and the max pooling layers to spatially decrease the feature maps to 4x4 pixels. From then, the top-down sequence up-samples the feature maps using nearest neighbour. To maintain the spatial information during up-sampling, skip connections are used between the bottom-up and top-down portions of the hourglass.

The network used is a pre-trained model of the original stacked hourglass network trained using the MPII [1] dataset.

5.1.3 Latent Pose Vector

After the final heatmap tensor is computed of the hourglass module in the first stage of the HyperStackNet, each predicted heatmap that is resulted from all eight of the hourglass modules are placed in a latent pose vector to be used for the final stage of the architecture. Each hourglass module output is of the size:

$$size(heatmap) = batchsize \times 16 \times 64 \times 64 \tag{5.1}$$

where the batchsize in the testing case is of size 3, the number of joints estimated is 16 and the image size is 64 by 64. By incorporating eight predicted heatmaps, the size of the latent pose vector (V_{lp}) is:

Table 5.1: Joint locations used in the HyperStackNet (see Figure 5.4).

Joint	Location	Joint	Location
Right Ankle	1	Head Top	10
Right Knee	2	Right Wrist	11
Right Hip	3	Right Elbow	12
Left Hip	4	Right Shoulder	13
Left Knee	5	Left Shoulder	14
Left Ankle	6	Left Elbow	15
Pelvis	7	Left Wrist	16
Thorax	8	Stick Upper	17
Upper Neck	9	Stick Lower	18

$$size(V_{lp}) = 8 \times 3 \times 16 \times 64 \times 64 \tag{5.2}$$

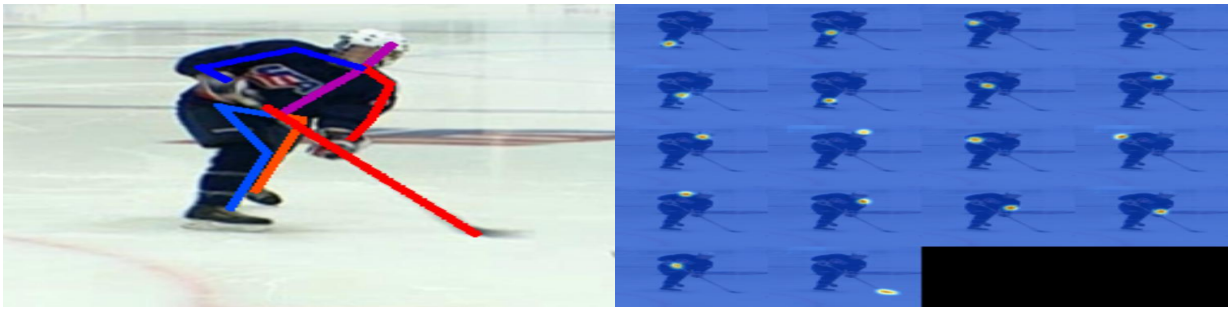
The heatmaps from each hourglass module were extracted at the end of their respective modules. By concatenating the pose information from the output of each hourglass module into a latent pose vector, the next stage of the network has information of successive refinement periods of the stacked hourglass network and can use the pose from each module to infer a more accurate pose.

5.1.4 Modified Stacked Hourglass Network

After the latent pose information concatenates to the latent pose vector a modified stacked hourglass network is then applied to determine the final pose estimation of 18 joints (16 human pose joints and two external joints representing the handheld object). The input of this stage of the network is the latent pose vector, while the output are heatmaps of the 18 joints. Identical to the stacked hourglass network, the final network prediction is produced by applying two consecutive rounds of 1x1 convolutions. Figure 5.4 and Table 5.1 show where the joints are located and the label of each joint. The final stage has weights that are randomly initialized and, similar to the original stacked hourglass network by Newell *et al.* [40], this latter half of the network also has an eight hourglass configuration.

The idea of the HyperStackNet is to infer more joint locations than what the original network can output. The intuition is to use existing joint information to learn additional

Figure 5.4: Corresponding positions of various joint locations (see Table 5.1).



(a) (b)

Figure 5.5: (a) Estimated pose overlayed on the image. (b) Heatmaps generated for the 18 joints. Note that the last row corresponds to the stick joints.

joints. Although the original stacked hourglass network was pretrained, the network as a whole is trained thereby finetuning the weights of the original stacked hourglass network from hockey data and improving the weights of the latter stacked hourglass module. In this application, the original network can infer 16 body joints, but not an additional two stick joints. Hence the importance of the latent pose vector which extracts 16 joint locations and is an input to the latter stacked hourglass network. By including the features from all eight modules in the first stage, one can continually infer information from each hourglass at different refinement periods. Therefore, by keeping all eight pose features and using that as an input latent feature to extract the final 18 joints, more information and therefore accuracy results. Figure 5.5 shows an example of the result of the HyperStackNet with the image accompanying heatmaps.

5.1.5 Transfer Learning Method

By incorporating the latent feature vector into the network, the overall architecture employs a method of transfer learning. The architecture uses latent features found within each module of the initial stacked hourglass network, with optimized weights associated with each hourglass module for estimating 16 joint locations, and uses that information to improve the estimation of the 16 joints on a hockey player in addition to two more points from the handheld object. As such, the method of transfer learning is novel because to the best knowledge of the authors, no research has been conducted thus far that extends a pose estimator from its original joint locations to that of more joints using latent pose as a feature with an extended network.

5.1.6 Dataset

The first half of the network is pre-trained using the MPII dataset, while the HyperStackNet as a whole was trained using the HARPE dataset [22]. The HARPE dataset consists of 886 images[1]. The dataset has annotated x and y coordinates of 18 joints, 16 of the body pose and two of the stick pose. The dataset consists of a single hockey player in their full equipment, on ice, with their hockey stick in the image. The players wore either blue or white jersey colors.

The dataset is partitioned randomly into three categories: training, validating, and testing. The data was partitioned into 70% training, 15% validation and 15% testing, giving the training set 620 images and validation and testing 133 images. The images in the dataset each shows the hockey player performing some type of skills (i.e., shooting, skating) or a drill (i.e., jumping, skate transitioning). Within those skills and drills, are active shots of a hockey player. The dataset is challenging because (1) similarity of the foreground (player) to the background (ice and boards), (2) bulky equipment occludes joints, and (3) motion blur prevents clear observability of the hockey stick. The dataset provides some unique challenges in pose estimation that differ from datasets found in previous publications.

5.1.7 Training Details

Training of the whole network was done with one epoch and with 8000 iterations per epoch. The training batch size was set to three. The input resolution of network is 256×256 pixels while the output resolution is 64×64 pixels. Images are preprocessed with .25 degrees of scale augmentation and 30 degree rotation of augmentation to reduce overfitting of the network. The network is trained using rmsprop for optimization with a learning rate of 2.5e-4. Batch normalization is implemented to improve training. The GPU used to train the network was an Nvidia Titan X.

5.2 Experimental Testing and Results

PCKh accuracy metric is used in the results because the metric reports the percentage of detections that fall within a normalized distance of the ground truth [1]. The PCKh

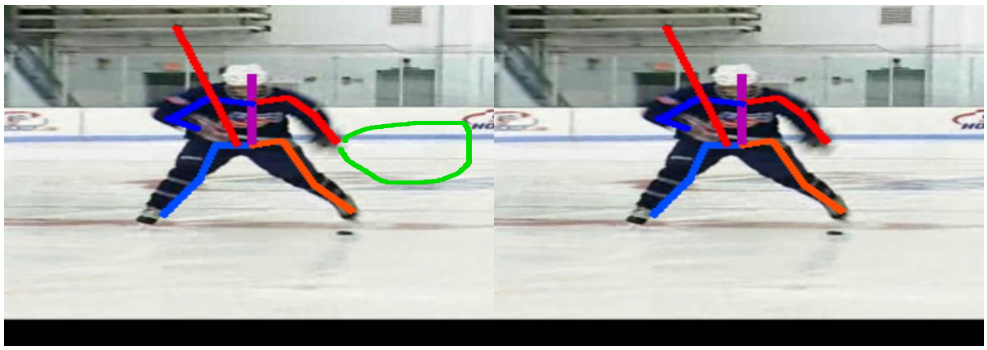
[1]Images collected to form the HARPE dataset were extracted from HOCKEY USA Skills and Drills DVD from Hockey USA, 2010

Table 5.2: PCKh comparison between the HyperStackNet and the (pre-trained) stacked hourglass network on the image test set for each joint location. In bold are the values that have the highest accuracy in each joint category. Averages are also computed for each network. From the results, the HyperStackNet outperforms the stacked hourglass in all joint categories.

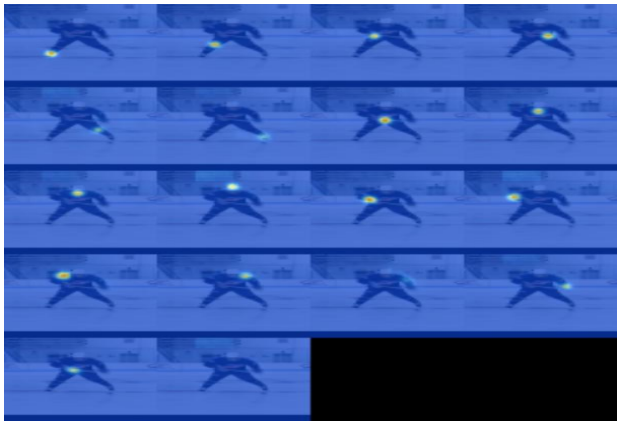
Joint	HyperStackNet(PCKh)	Stacked Hourglass (PCKh)
Head	**100**	91.9
Shoulder	**99.6**	90.6
Elbow	**99.2**	80.4
Wrist	**98.4**	78.9
Hip	**99.2**	86.4
Ankle	**100**	91.9
Knee	**99.6**	93.6
Stick Upper	**100**	N/A
Stick Lower	**87.3**	N/A
Average	**98.8**	85.4

measure for individual joints of HyperStackNet and the stacked hourglass network using the same image test set are shown in Table 5.2. From the table, the HyperStackNet has relatively low PCKh values for wrist and elbow joints due to the motion blur effect during player actions. Perfect accuracy is obtained for head and ankle joints. The PCKh value for the top of the stick is accurately estimated because the top of the stick tends to be held consistently in the glove of the hockey player. The PCKh value for the lower end of the stick, however, is quite low, the reason for this can be attributed to the excessive amount of motion blur present in the stick movement as shown in Figure 5.6. Other failure cases include the presence of self occlusion shown in Figure 5.7.

To understand visually how the HyperStackNet compares to the stacked hourglass network, Figure 5.8 provides three test images from the HARPE dataset and displays their respective pose estimates of each architecture; each section (i.e., left arm, right arm, torso, left leg, right leg and stick) is colour coded and overlayed on the image. From the figure, the HyperStackNet correctly identifies almost every joint location on the hockey player and on the stick, whereas, the stacked hourglass network, as apparent in the first comparison image in Figure 5.8, does not correctly identify a portion of joint locations. At times, the stacked hourglass network does not correctly identify the left arm, or fails to correctly



(a) (b)



(c)

Figure 5.6: (a) A typical failure case which is due to motion blur plus look-alike. Note that even for human annotators, the stick joint locations are difficult to locate. (b) Corresponding pose prediction. (c) Corresponding heatmap. Note that the second heatmap corresponding to the stick is not present due to motion blur.

(a) (b)

Figure 5.7: (a) A typical failure case where the stick is not identified due to self occlusion (b) Corresponding heatmap of the image on left.

identify an ankle. What is most apparent is that the stacked hourglass network is not viewpoint invariant in contrast to the HyperStackNet that accurately identifies pose on hockey players where the viewpoint is angled steeply down onto the ice as shown in the first image comparison of Figure 5.8.

Compared to the pre-trained stacked hourglass network, the HyperStackNet trained using transfer learning, a larger dataset and one epoch achieves a better average accuracy of joints by a margin of 13.4 percent as shown in Table II. The three lowest PCKh values for the stacked hourglass network were the wrist, elbow and hip, which, the PCKh value significantly improved by a margin of 19.5%, 18.8% and 12.8% when using the HyperStack-Net.

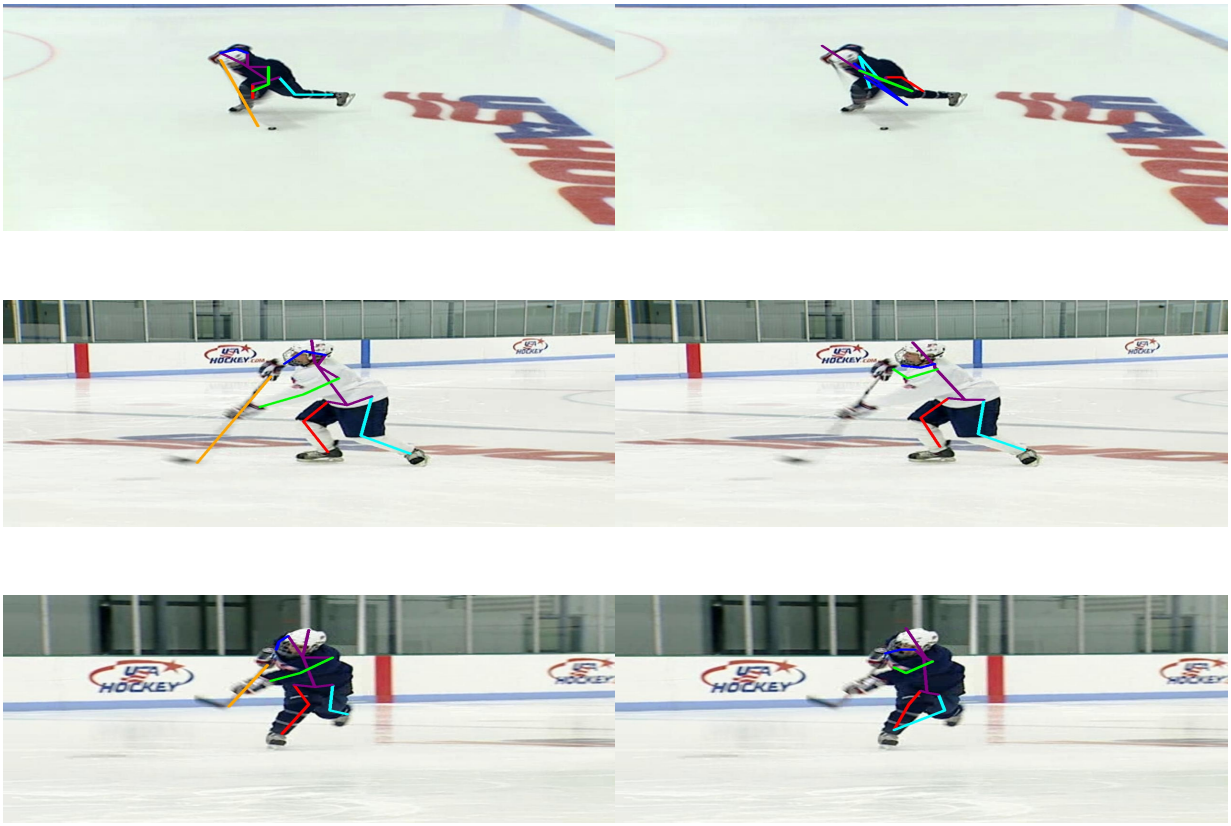


Figure 5.8: Comparison between some test results of HyperStackNet model (left) and the original pre-trained stacked hourglass network (right). Evidently, the HyperStackNet model does better at wrists, ankle and pelvis joints.

Chapter 6

Action Recognition using Pose Estimation Features

In the previous chapter, the efficacy of applying pose estimation to ice hockey was validated thus demonstrating that pose estimation provides valuable pieces of information that can potentially help players and coaches. In addition, the use of action recognition in computer vision is another method to help.

Action recognition in computer vision is an important and popular problem in the application of analyzing sport videos. Action recognition provides a benefit to coaches, analysts and spectators by providing content for coaches and analysts to evaluate player performance and for spectators to view content. Although only a limited amount of action recognition or even computer vision research has been done in the field of hockey, action recognition can be applied to hockey analytics to analyze characteristics of hockey players and teams.

In this article, videos captured by a single camera is employed, and a convolutional neural network (CNN), called Action Recognition Hourglass Network (ARHN), is introduced that extracts pose features from hockey images and videos and utilizes them for action recognition. Although the use of depth sensors can be employed, that method is expensive and the data gathered is too noisy. Action recognition for broadcast videos, although more challenging, is more desirable and more realistic.

A dataset of annotated hockey images was generated; to the best of our knowledge, there is no publicly available benchmark hockey dataset for action recognition and pose estimation. In this dataset, video frames of hockey players performing four types of activities (namely, cross-overs, straight skating, pre-shot, and post-shot) are labeled and body

joint locations are annotated.

The main contributions include the following: 1) ARHN architecture overview which includes the general framework and discussion on the pieces of the architecture (Section 6.1), and 2) experimental testing and results (Section 6.2). This research focuses on utilizing pose information for action recognition and does not employ temporal features for two reasons. First, a player's pose, as a static feature, is a strong clue for action recognition, and, second, incorporating temporal information like motion descriptors arises the need for a much bigger dataset to be used for training a deep structure that incorporates temporal information.

6.1 Architectural Overview

As indicated earlier, this research involves the development, implementation, and testing of a new method to perform action recognition. This method is applied to recognizing actions of ice hockey players as an example. The ARHN uses features based on latent pose estimation to estimate action recognition using single video frames of hockey players. An overview of the framework is demonstrated in Figure 6.1 and is described in Section 6.2.2.

6.1.1 Proposed General Framework

The proposed action recognition framework in video is illustrated in Figure 6.1. As shown in Figure 6.1, a hockey video-segment is converted to a sequence of frames. In each frame a player is tracked, and a coordinate of their body center is determined. Next, the frame resolution is adjusted to the proper input size (i.e., 720×1280) of the network. Then a region of interest (with size 250×250), centered at a player's body-center is cropped from the image and is given to the ARHN network. The network, by finding heatmaps (where each heatmap corresponds to the predicted probability of a joint's presence at each image-pixel [40]), generates the pose estimation which is then used to estimate player action. Four different types of hockey player actions are considered which are: cross-over, straight skating, pre-shot, and post-shot. Details of the ARHN structure are presented and discussed in the next subsection.

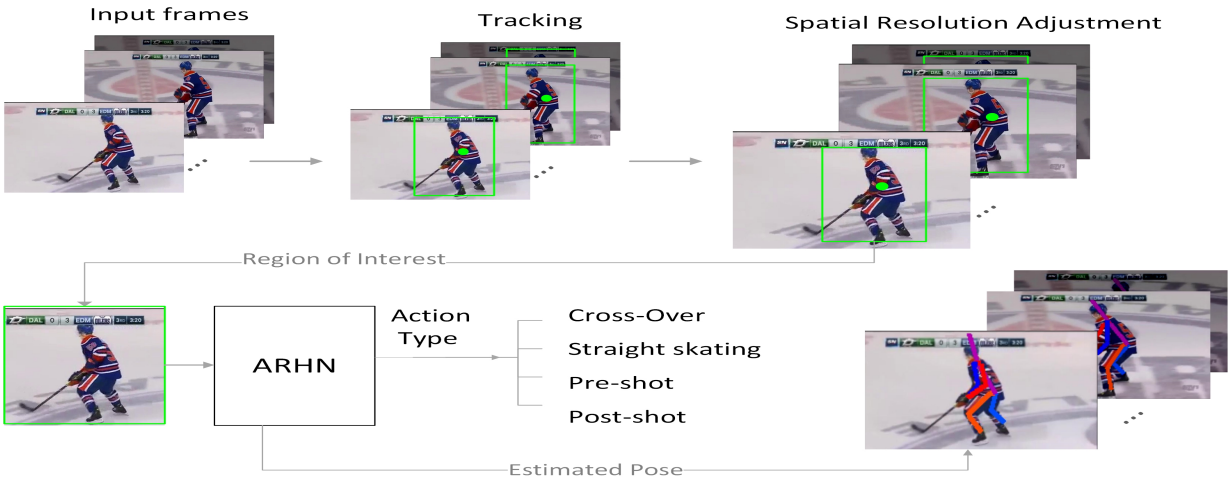


Figure 6.1: Implemented framework for hockey action recognition through pose estimation for hockey images/video frames. The framework begins by extracting video frames as input, the body-center of the player is determined using tracking means, then the image size is scaled and fed into the network. The network then classifies the action and overlays the estimated pose on the image.

6.1.2 Network Architecture

The general structure of the ARHN is presented in Figure 6.2 and broken into three components. The first component is the stacked hourglass network [40], which inputs the raw image and generates a set of heatmaps that defines the pose as the latent feature. The second component of the network is the latent feature transformer that receives the latent features and transforms them to a common frame of reference. The third component is the action recognition classifier which is composed of six fully-connected layers and classifies a hockey player' s action type. Sequencing these three parts, as shown in Figure 6.2, constructs the ARHN network as a unified deep structure.

The pose estimator component implicitly learns the pose of a hockey player through the use of a generated set of statistical probability heatmaps that identify the joint locations of a hockey player in a still image. Then, the latent feature transformer scales and shifts the learned pose, forming a feature vector. The fully connected layers in the third component perform the action recognition task.

To understand the ARHN, a brief overview of the original hourglass network, in con-

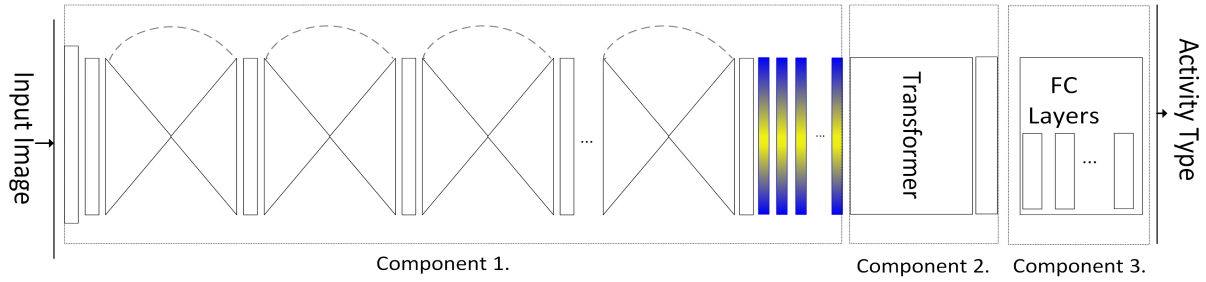


Figure 6.2: Proposed ARHN for action recognition identifying the three components. Component 1: pose estimation using an hourglass network. Component 2: feature transformation to transform poses into a common frame of reference Component 3: action recognition represented by fully connected layers.

junction with a description of latent feature transformer and fully-connected layers are provided respectively in Subsections 6.2.3, 6.2.4, and 6.2.5.

6.1.3 Latent Pose Estimation via Stacked Hourglass Network

The stacked hourglass network is a deep convolutional network architecture composed of multiple hourglass modules put together in series [40]. Each hourglass module has convolutional, max-pooling, and up-sampling layers as its basic elements to realize a bottom-up, top-down mechanism for generating feature maps. In bottom-up sequence, successive convolution and max pooling layers are engaged to bring the resolution of feature maps to 4x4 pixels. In the top-down sequence, feature maps are up-sampled using nearest neighbor. The major elements of this architecture are skip connections between bottom-up and top-down sections of the hourglass, which are shown by dashed arches in Figure 6.2. These skip connections preserve information of high resolution feature maps, in the first section of network, to be combined with features of other scales, in the second section. The hourglass network generates a set of 16 statistical heatmaps. Figure 6.3 provides instances of some heat maps for the right ankle, right knee and right hip of a hockey player. These heatmaps actually form the latent pose features for the ARHN network.

Figure 6.3: Statistical heat maps demonstrating the probability of the location of the right ankle, right knee, and right hip (left to right) for a hockey player.

6.1.4 Latent Feature Transformer

The second component is a feature transformer that transforms pose heatmaps to a common frame of reference by performing spatial translation and scaling in 2-D plane. The location of a peak in a heatmap gives the predicted coordinate of a joint for the input image. A specific constellation of joints (i.e., geometrical arrangement of a set of joints) shows the pose of a player. A player's pose potentially should represent a particular type of action, that is performed; typical poses for four types of actions in hockey are indicated in Figure 6.4. However, poses that represent the same action type can vary significantly in the joint position, orientation, and sizes. To generate a more consistent representation for poses, referred to as canonical poses, the feature transformer is used. This component generates the canonical poses from the heatmaps to generate a better pose representation to be used as input into the action recognition component.

The latent feature transformer, is demonstrated in Figure 6.5(a). All joint coordinates are shifted with respect to a point defined as the body center (x_0, y_0), namely, the point halfway between the thorax and pelvis keypoints indicated by "O" on the stickmen in Figure 6.4. Joint coordinates are scaled by scaling ratio S as per Eq. (6.1). S is the ratio of the average head size of players in all training images (N) and H_n is the head size of the player in the n^{th} image. Head size is the distance between the "head top" and "upper neck" keypoints (Fig. 4).

$$S = \frac{\sum_{n=1}^{N} H_n}{H_n} \tag{6.1}$$

As shown in Figure 6.5, besides transformed coordinates (i.e., $[x_i, y_i]^T$) of 16 body-joints, angles (α_j) between some joints are also calculated. Angles which are computed are between (right & left)- shoulder, (right & left)-elbow, (right & left)-hip, and (right & left)-

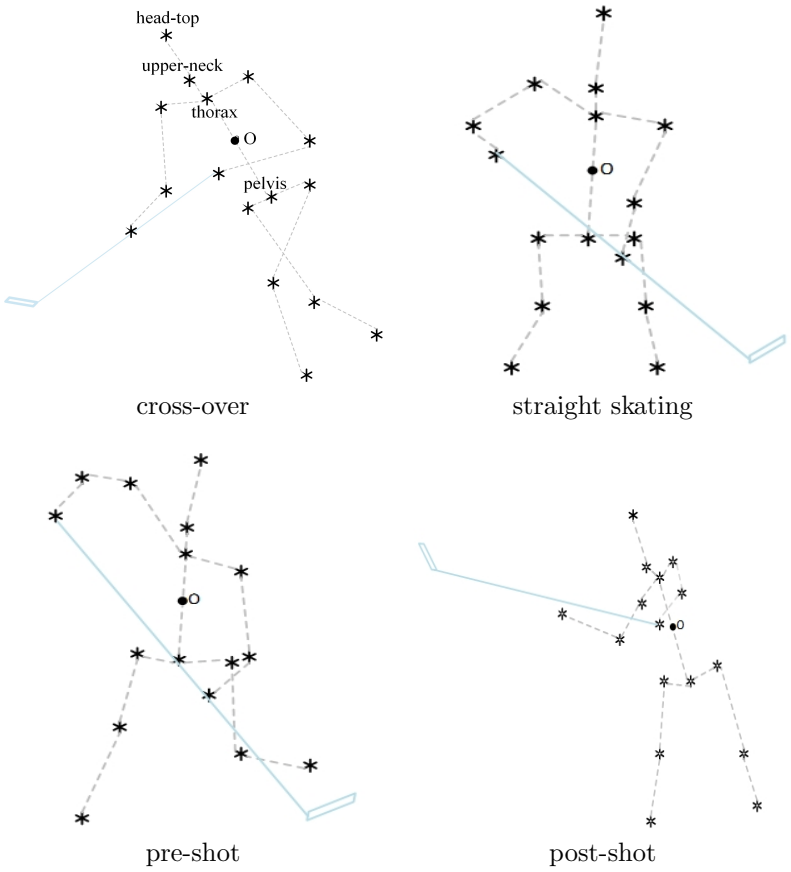


Figure 6.4: Typical poses for 4 different actions of a hockey player

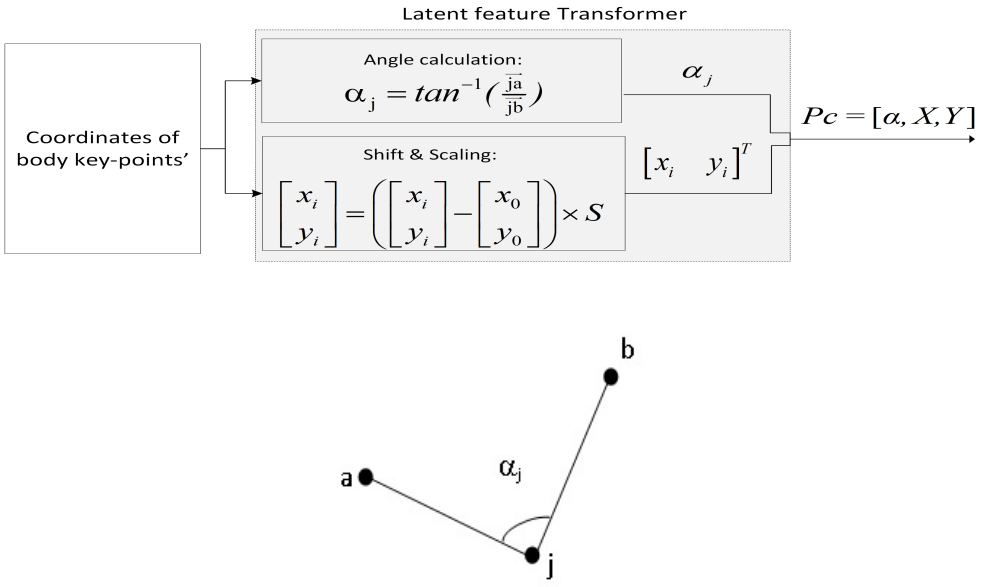


Figure 6.5: (a) Latent feature transformer, which generates canonical pose vector by shifting and scaling joint coordinates and computing the joint angles. (b) Angle of joint "j" (i.e., α_j) is the smaller angle between vectors ja and jb.

knee joints. The output of the latent feature transformer is a 40-dimensional vector named the canonical pose (p_c) given in Eq. (6.2). This vector is formed by concatenation of joint angles and transformed keypoint coordinates. The canonical pose is the feature that is next evaluated by the third component of the ARHN to perform action recognition.

$$p_c = \begin{bmatrix} \alpha & X & Y \end{bmatrix} \tag{6.2}$$

$$X = \begin{bmatrix} x_1, & x_2, & ..., & x_i, & ... & x_{16} \end{bmatrix}$$
$$, Y = \begin{bmatrix} y_1, & y_2, & ..., & y_i, & ... & y_{16} \end{bmatrix}$$
$$, \alpha = \begin{bmatrix} \alpha_1, & \alpha_2, & ..., & \alpha_j, & ... & \alpha_8 \end{bmatrix}$$

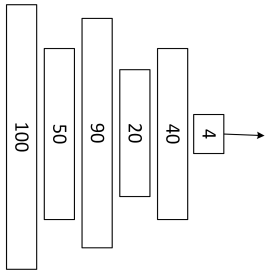


Figure 6.6: Action recognition component consisting of 6 fully connected layers beginning with a layer of 100 neurons to 50, 90, 20, 40 and ending with a fully connected layer of 4 to perform action recognition.

6.1.5 Action Recognition Component

The last component of the network is illustrated in Figure 6.6. This component is composed of six fully connected layers to recognize activities. The fully connected layers receive the 40-dimensional feature vector from the latent feature transformer, passing it through five fully connected layers with sigmoid activation functions and a final layer of four neurons with a hard-limit function to recognize one of the four types of activities for the input image. The number of neurons in each layer is indicated in Figure 6.6. Note that the number of layers and the number of neurons per layer are determined empirically.

6.2 Experimental Testing and Results

Experiments that are conducted here assess the performance of ARHN for action recognition in the context of hockey. Both visual and numerical evaluations are provided.

6.2.1 Dataset Preparation

In machine learning, and particularly deep learning problems, having access to a proper dataset is a crucial requirement. Deep networks generally use supervised or semi-supervised algorithms for learning, which heavily rely on annotated data during training. Deep networks are designed to extract information from raw input data, therefore, performance relies on the data samples presented to deep networks. If the provided data are not representative of the problem, or the number of training samples is limited, the machine learning

method fails to properly tune its parameters and cannot provide an accurate model for solving the problem.

In the context of hockey, no standard set of annotated hockey images for pose estimation or action recognition is available. Therefore, in this work, a dataset, named HARPE, has been collected and annotated for this purpose.

- Video segments are captured from a set of hockey videos and converted to video frames.
- Video frames are categorized into classes based on the four hockey actions: cross-overs, straight skating, pre-shot, and post-shot.
- Very low quality frames, and frames unrepresentative of classes, are manually detected and discarded.
- Spatial resolution of each frame is adjusted to the proper size for delivering to the network i.e., 720×1280.
- A hockey player is tracked in all frames to determine his body center in pixel coordinates.
- For each frame, the positions of 16 body joints (Table 6.1) for the player of interest is annotated and the action type is labeled.
- The two ends of the hockey sticks are also annotated in each frame for future use.

In summary, keypoints are annotated in 887 frames with an associated action label. The dataset has 1676 frames of cross-overs, 271 frames of straight skating, 245 frames of pre-shooting, and 203 frames of post-shooting. The dataset is planned to be publicly available.

6.2.2 Accuracy of Action Recognition

To evaluate the accuracy of action recognition the images are randomly divided into three groups: 70% training, 15% validation, and 15% validation. Training images are passed through the ARHN network and the parameters of network are tuned accordingly. Due to the limited size of the provided data, parameters of hourglass layers are hardly affected (weights of hourglass layer are pre-trained by general human poses on MPII dataset [1]),

Table 6.1: List of annotated key-points for each frame.

#	Key-point	#	Key-point
1	Right ankle	10	Head top
2	Right knee	11	Right wrist
3	Right hip	12	Right elbow
4	Left hip	13	Right shoulder
5	Left knee	14	Left shoulder
6	Left ankle	15	Left elbow
7	Pelvis	16	Left wrist
8	Thorax	17	top of stick
9	Upper neck	18	end of stick

while parameters of the fully connected layers are the ones that are mainly learned during the training phase. This process has been repeated with fifteen randomly selected groups, and the average performance of ARHN network for action recognition is reported. The 70/15/15 splitting of data and averaging over 15 runs, is validated in Subsection 6.3.3.

For this purpose, precision and recall rates for training, testing, and validation images are computed and provided respectively in Tables 6.2, 6.3, and 6.4 for each of the four class types; where 1 represents cross-overs, 2 represents straight skating, 3 represents pre-shot, and 4 represents post-shot.

The precision and recall rates of Table 6.4, for the test data, show that the network has precision of about 65% for each class. However, in many cases, a hockey players' pose in cross-over and straight skating (the two first classes) are quite similar to each other. It is also the case for pre-shot and post-shot (the two last classes). For each type of action, some examples of correctly classified and misclassified images are illustrated in Figures 6.7 and 6.8. In Figure 6.7, all images follow the typical action poses shown in Fig. 6.4, so they are all correctly classified by the ARHN. In contrast, images of Figure 6.8 are all misclassified because they deviate from their true class and mimic a different class. Considering player poses, misclassification of Figure 6.8 by ARHN can be justified. This subject is further investigated in the next experiment.

Table 6.2: Performance of ARHN for training.

Class #	1	2	3	4
Precision (%)	68.3	7.18	75.9	79.5
Recall (%)	68.6	74.1	77.0	73.0

Table 6.3: Performance of ARHN for validation.

Class #	1	2	3	4
Precision (%)	64.5	68.8	72.6	64.1
Recall (%)	69.5	68.9	68.4	64.1

6.2.3 Effect of Merging Classes

The purpose of this experiment is to show that by merging similar classes, accuracy of classification can be improved. In Figure 6.9, a confusion matrix for one run on training data is provided.

The confusion matrix in Figure 6.9 shows that most misclassifications occur between classes 3 and 4 (pre-shot and post-shot), as well as classes 1 and 2 (cross-over and straight skating); shooting classes are clearly distinct from the skating classes. Therefore, in Table 6.5, the effect of merging similar classes on accuracy of action recognition is investigated. Mean classification accuracy averaged over 15 and then 1000 runs are reported for three different testing conditions.

In the first test none of the classes are merged together. In the second test the two last classes (i.e., pre-shot and post-shot) are merged together. Finally, in the third test

Table 6.4: Performance of ARHN for testing.

Class #	1	2	3	4
Precision (%)	61.7	67.0	68.3	63.1
Recall (%)	61.7	67.0	68.1	63.1

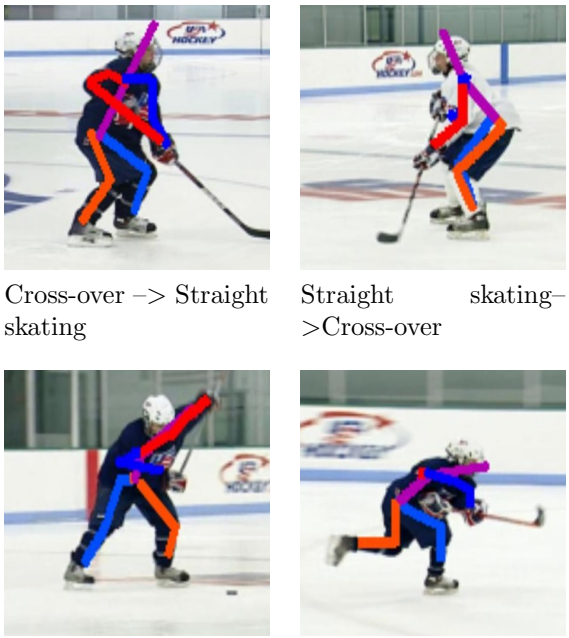
Cross-over Straight skating

Pre-shot Post-shot

Figure 6.7: Examples of activities, correctly classified.

Table 6.5: Accuracy of action recognition over 15 and 1000 runs for three testing conditions: evaluating classes 1,2,3 and 4 as separate classes, evaluating class 1 and 2 separately with classes 3 and 4 as a single class, and evaluating classes 1 and 2 as one separate class and classes 3 and 4 as another class.

Class Indices	1,2,3,4	1,2,(3,4)	(1,2),(3,4)
Mean 15 runs(%)	65.14	71.13	78.32
Mean 1000 runs(%)	65.47	69.08	78.49
Variance 1000 runs	0.0064	0.0043	0.0030



Cross-over –> Straight skating

Straight skating–>Cross-over

Pre-shot–>Post-shot

Post-shot–>Pre-shot

Figure 6.8: Examples of missclassified activities. In each case the true class-type followed by the predicted class-type are shown under the image in question.

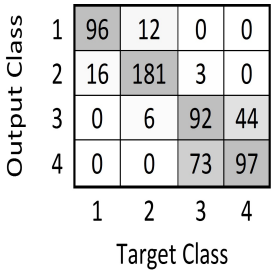


Figure 6.9: Confusion matrix of action recognition for one run.

the first two classes (i.e., cross-over and straight skating) are also combined. Accuracy of recognition for each of these testing conditions are provided in the three columns of Table 6.5. Table 6.5 demonstrates that by unifying similar classes, the mean accuracy increases. Also, Table 6.5 shows that the mean accuracy over 15 runs is close to the mean accuracy over 1000 runs. The low variance over 1000 runs validates that fewer runs (e.g., 15) should be sufficient for representative results. The result of this test for 1000 runs are also demonstrated in the form of histograms in Figure 6.10 **??**or each of the three testing conditions. The histograms show that by merging the classes, mean accuracy increases and variance decreases resulting in the histogram to look more concentrated.

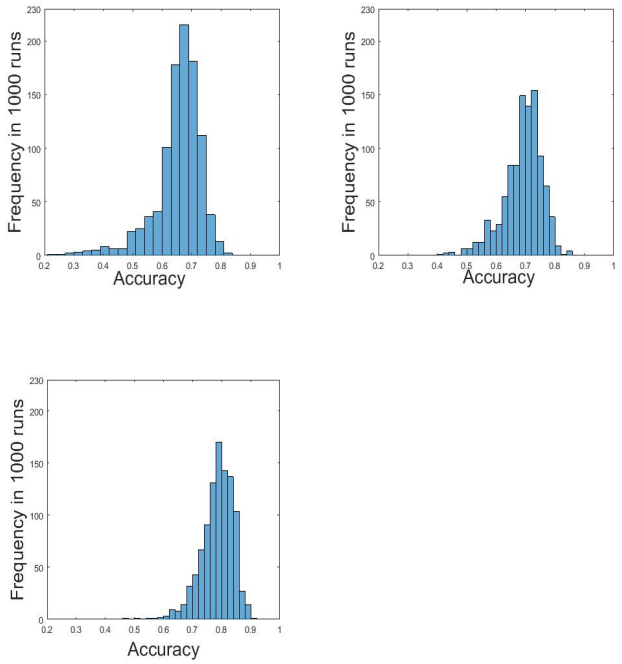


Figure 6.10: Histograms of accuracy over 1000 runs for random selection of test samples. (a) all 4 classes as separate classes (b) classes 3 and 4 acting as one class and class 1 and 2 as separate classes and (c) classes 1 and 2 are a single class as well as classes 3 and 4.

Chapter 7

Conclusions

In this work, an automated method to determine the pose of a hockey player with and without a hockey stick from broadcast game video in addition to performing action recognition via pose is achieved. In this dissertation, experiments using the stacked hourglass network proved the efficacy of deep learning in challenging scenarios like hockey videos, where severe occlusions (i.e., bulky clothing and high speed of players due to skating), exist for pose estimation. In addition, the successful numerical and visual outcomes of the HyperStackNet prove that the employed convolutional neural network, along with using hand-held objects further improve the estimated pose of a hockey player. The deep structure called ARHN network is designed and implemented which also successfully performs action recognition in the sport of hockey using latent pose estimation features.

The results of this research provide a contribution to hockey analytics. By estimating the pose of a player through a convolutional neural network, coaches and analysts can utilize the joint data received to evaluate a hockey player based on their capability as a player rather than based on the effects of a hockey player's performance.

7.1 Potential for Future Research

The researched presented in this dissertation provides a basis for future research in pose estimation and action recognition. There are two potential research topics that can build upon the research presented in this dissertation:

- Temporal pose estimation

- Temporal action recognition

Within temporal pose estimation and temporal action recognition, many avenues of research can be explored. Modifying the neural network to incorporate temporal features is the first step. One method could include testing long-short term memory (LSTM) networks. In pose estimation, experimenting on the use of extra-join location in temporal pose estimation may also be of interest; by knowing the movement of a hand-held object, one can infer the joint locations holding the object. In action recognition, one can refine the action recognition pipeline by including other latent features which may include joint locations, or angle references of joints with respect to a key coordinate.

The research presented provided experimental results in pose estimation and action recognition in hockey using still images. The system proved to be effective in the field of hockey, however, some challenges still arose. For example, some joint locations due to occlusion were inaccurate and for action recognition some actions were very similar to each other. To improve occlusion in pose estimation and similarity in action recognition, the employment of temporal features would be solve these problems.

7.2 Thesis Applicability

The experiments and results in this thesis has provided a practical solution to pose estimation and action recognition using convolutional neural networks. From the experimental results, pose estimation and action recognition using the stacked hourglass, HyperStackNet and the ARHN are excellent architectures for performing pose estimation and action recognition on still images. The experiments that these networks can effectively perform pose estimation and action recognition tasks in challenging scenarios that include occlusion, bulky equipment and background/foreground similarity. This work can therefore be applied to other forms of fields that need accurate information which include, health care and security.

7.3 Thesis Impact

There are many contributions from the dissertation. This dissertation introduces pose estimation in hockey using broadcast images while also introducing a novel idea of modelling pose of a hockey stick. In addition, the extension of pose estimation in hockey to action recognition was also discussed. Experimental results in pose estimation, modeling of pose

of the hockey stick and the extension of hockey pose estimation in action recognition was also established.

APPENDICES

Appendix A

Code Base Pose Estimation

Section 4 and Section 5 detail code that is used in hockey pose estimation with 16 body locations in one section and with body joint locations in addition to 2 hockey stick locations. Code used for Section 4 are from the stacked hourglass network by Newell *et al.* [40] and the code can be found in the Michigan Vision Learning Lab's github account. The demo code can be found in the following url: https://github.com/umich-vl/pose-hg-demo. The test code can be found in the following url: https://github.com/umich-vl/pose-hg-train.

Experiments used for Section 5 use a modified version of the stacked hourglass network that can be found on this author's github account at https://github.com/neherh/HyperStackNet. The following sections provide a code summary of the files that were modified from the original stacked hourglass network for training and testing. Files modified in the train code include hg.lua, harpe.lua, and model.lua. In the demo code, files modified include main.lua, util.lua, and img.lua. The following sections provide the training code (Section A.1) and the demo code (Section A.2) and explanations of each modifications.

A.1 HyperStackNet Training Code

The following subsections are explanations of the various lua language files modified for the HyperStackNet training.

A.1.1 harpe.lua

This file was created and used to load the HARPE dataset for training.

```
local M = {}
Dataset = torch.class('pose.Dataset',M)

function Dataset:__init()
    self.nJoints = 18
    self.accIdxs = {1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18}
    self.flipRef = {{1,6},   {2,5},   {3,4},
                    {11,16}, {12,15}, {13,14}}
    -- Pairs of joints for drawing skeleton
    self.skeletonRef = {{1,2,1},    {2,3,1},    {3,7,1},
                        {4,5,2},    {4,7,2},    {5,6,2},
                        {7,9,0},    {9,10,0},
                        {13,9,3},   {11,12,3},  {12,13,3},
                        {11,17,1},  {17,18,1},
                        {14,9,4},   {14,15,4},  {15,16,4}}

    local annot = {}
    local tags = {'index','person','imgname','part','center','scale',
                  'normalize','torsoangle','visible','multi','istrain'}
    local a = hdf5.open(paths.concat(projectDir,'data/harpe/annot_corrected.h5'),'r')
    for _,tag in ipairs(tags) do annot[tag] = a:read(tag):all() end
    a:close()
    annot.index:add(1)
    annot.person:add(1)
    annot.part:add(1)

    -- Index reference
    if not opt.idxRef then
        local allIdxs = torch.range(1,annot.index:size(1))
        opt.idxRef = {}
        opt.idxRef.test = allIdxs[annot.istrain:eq(0)]
        opt.idxRef.train = allIdxs[annot.istrain:eq(1)]

        if not opt.randomValid then
```

```
            -- Use same validation set as used in our paper (and same as Tompson et a
            tmpAnnot = annot.index:cat(annot.person, 2):long()

            tmpAnnot:add(-1)

            local validAnnot = hdf5.open(paths.concat(projectDir, 'data/harpe/annot/val
            local tmpValid = validAnnot:read('index'):all():cat(validAnnot:read('perso
            opt.idxRef.valid = torch.zeros(tmpValid:size(1))
            opt.nValidImgs = opt.idxRef.valid:size(1)
            opt.idxRef.train = torch.zeros(opt.idxRef.train:size(1) - opt.nValidImgs)
            -- Loop through to get proper index values
            local validCount = 1
            local trainCount = 1
            for i = 1,annot.index:size(1) do
                if validCount <= tmpValid:size(1) and tmpAnnot[i]:equal(tmpValid[valid
                    opt.idxRef.valid[validCount] = i
                    validCount = validCount + 1
                elseif annot.istrain[i] == 1 then
                    opt.idxRef.train[trainCount] = i
                    trainCount = trainCount + 1
                end
            end
        else
            -- Set up random training/validation split
            local perm = torch.randperm(opt.idxRef.train:size(1)):long()
            opt.idxRef.valid = opt.idxRef.train:index(1, perm:sub(1,opt.nValidImgs))
            opt.idxRef.train = opt.idxRef.train:index(1, perm:sub(opt.nValidImgs+1,-1))
        end

        torch.save(opt.save .. '/options.t7', opt)
    end

    self.annot = annot
    self.nsamples = {train=opt.idxRef.train:numel(),
                     valid=opt.idxRef.valid:numel(),
                     test=opt.idxRef.test:numel()}

    -- For final predictions
```

```
73      opt.testIters = self.nsamples.test
74      opt.testBatch = 1
75  end
76
77  function Dataset:size(set)
78      return self.nsamples[set]
79  end
80
81  function Dataset:getPath(idx)
82      return paths.concat(opt.dataDir,'images',ffi.string(self.annot.imgname[idx]:char()
83  end
84
85  function Dataset:loadImage(idx)
86      return image.load(self:getPath(idx))
87  end
88
89  function Dataset:getPartInfo(idx)
90      local pts = self.annot.part[idx]:clone()
91      local c = self.annot.center[idx]:clone()
92      local s = self.annot.scale[idx]
93      -- Small adjustment so cropping is less likely to take feet out
94      --c[2] = c[2] + 15 * s
95      s = s * 1.8
96      return pts, c, s
97  end
98
99  function Dataset:normalize(idx)
100     return self.annot.normalize[idx]
101 end
102
103 return M.Dataset
```

A.1.2 hg.lua

The function createModel in lines 35 - 81 are added lines of code which creates a new model based on the input of 16 concatenated heatmaps rather than an image.

```
paths.dofile('layers/Residual.lua')


local function hourglass(n, f, inp)
    -- Upper branch
    local up1 = inp
    for i = 1,opt.nModules do up1 = Residual(f,f)(up1) end

    -- Lower branch
    local low1 = nnlib.SpatialMaxPooling(2,2,2,2)(inp)
    for i = 1,opt.nModules do low1 = Residual(f,f)(low1) end
    local low2

    if n > 1 then low2 = hourglass(n-1,f,low1)
    else
        low2 = low1
        for i = 1,opt.nModules do low2 = Residual(f,f)(low2) end
    end

    local low3 = low2
    for i = 1,opt.nModules do low3 = Residual(f,f)(low3) end
    local up2 = nn.SpatialUpSamplingNearest(2)(low3)

    -- Bring two branches together
    return nn.CAddTable()({up1,up2})
end

local function lin(numIn,numOut,inp)
    -- Apply 1x1 convolution, stride 1, no padding
    local l = nnlib.SpatialConvolution(numIn,numOut,1,1,1,1,0,0)(inp)
    return nnlib.ReLU(true)(nn.SpatialBatchNormalization(numOut)(l))
end


function createModel(inp)
     print('==> In create model ')
     if opt.trainType  == 'inception' then
         pretrained = inp
```

```
39            inp = nn.Identity()()
40            mapping = nnlib.SpatialConvolution(8,opt.nFeats,1,1,1,1,0,0)(inp)
41         else
42            inp = inp()
43            summed_heatmaps = nn.CAddTable()(inp)
44            mapping = nnlib.SpatialConvolution(ref.nOutChannels,opt.nFeats,1,1,1,1,0,0)(su
45         end
46
47
48         local out = {}
49         local inter = mapping
50
51        for i = 1,opt.nStack do
52            local hg = hourglass(4,opt.nFeats,inter)
53
54            --Residual layers at output resolution
55            local ll = hg
56            for j = 1,opt.nModules do ll = Residual(opt.nFeats,opt.nFeats)(ll) end
57              --Linear layer to produce first set of predictions
58            ll = lin(opt.nFeats,opt.nFeats,ll)
59
60            -- Predicted heatmaps
61            local tmpOut = nnlib.SpatialConvolution(opt.nFeats,ref.nOutChannels,1,1,1,1,0,
62            table.insert(out,tmpOut)
63
64             -- Add predictions back
65            if i < opt.nStack then
66                local ll_ = nnlib.SpatialConvolution(opt.nFeats,opt.nFeats,1,1,1,1,0,0)(ll
67                local tmpOut_ = nnlib.SpatialConvolution(ref.nOutChannels,opt.nFeats,1,1,1
68                inter = nn.CAddTable()({inter, ll_, tmpOut_})
69            end
70         end
71
72          --Final model
73         local model = nn.gModule({inp}, out)
74
75          if opt.trainType  == 'inception' then
76            return pretrained , model
```

```
77       else
78          return model
79        end
80
81  end
82
83
84  function createModelFresh()
85      print('==> Creating a fresh one without pretraining ')
86
87
88  -- print("In cREATE MODEL")
89      local inp = nn.Identity()()
90      -- Initial processing of the image
91      local cnv1_ = nnlib.SpatialConvolution(3,64,7,7,2,2,3,3)(inp)           -- 128
92      local cnv1 = nnlib.ReLU(true)(nn.SpatialBatchNormalization(64)(cnv1_))
93      local r1 = Residual(64,128)(cnv1)
94      local pool = nnlib.SpatialMaxPooling(2,2,2,2)(r1)                       -- 64
95      local r4 = Residual(128,128)(pool)
96      local r5 = Residual(128,opt.nFeats)(r4)
97
98      local out = {}
99      local inter = r5
100
101     for i = 1,opt.nStack do
102         local hg = hourglass(4,opt.nFeats,inter)
103
104         -- Residual layers at output resolution
105         local ll = hg
106         for j = 1,opt.nModules do ll = Residual(opt.nFeats,opt.nFeats)(ll) end
107         -- Linear layer to produce first set of predictions
108         ll = lin(opt.nFeats,opt.nFeats,ll)
109
110         -- Predicted heatmaps
111         local tmpOut = nnlib.SpatialConvolution(opt.nFeats,ref.nOutChannels,1,1,1,1,0,(
112         table.insert(out,tmpOut)
113
114         -- Add predictions back
```

```
115         if i < opt.nStack then
116             local ll_ = nnlib.SpatialConvolution(opt.nFeats,opt.nFeats,1,1,1,1,0,0)(ll)
117             local tmpOut_ = nnlib.SpatialConvolution(ref.nOutChannels,opt.nFeats,1,1,1,
118             inter = nn.CAddTable()({inter, ll_, tmpOut_})
119         end
120     end
121
122     -- Final model
123     local model = nn.gModule({inp}, out)
124
125     return model
126
127 end
```

A.1.3 model.lua

Lines 18-19 were added pieces of code that loads a pre-trained network of the stacked hourglass network, adds up the 16 heatmaps in hg.lua createModel function, builds a new stacked hourglass on network on top of the pre-trained model and then jointly trains the whole model.

```
1  --- Load up network model or initialize from scratch
2  --require('mobdebug').start()
3  paths.dofile('models/' .. opt.netType .. '.lua')
4
5
6  -- Continuing an experiment where it left off
7  if opt.continue or opt.branch ~= 'none' then
8      local prevModel = opt.load .. '/final_model.t7'
9      print('==> Loading model from: ' .. prevModel)
10     model = torch.load(prevModel)
11
12
13 -- Or a path to previously trained model is provided
14 elseif opt.loadModel ~= 'none' then
15     assert(paths.filep(opt.loadModel), 'File not found: ' .. opt.loadModel)
16     print('==> Loading model from!!!!!!: ' .. opt.loadModel)
```

```
    pretrained_8 = torch.load(opt.loadModel)
    pretrained_12 = torch.load('/home/neherh/models_mpii/model_56.t7')
     model = createModel(pretrained)

      model = modelTrained

-- Or we're starting fresh
else
    print('==> Creating model from file: models/' .. opt.netType .. '.lua')
    print(modelArgs)
    model = createModelFresh(modelArgs)
end

-- Criterion (can be set in the opt.task file as well)
if not criterion then
    criterion = nn[opt.crit .. 'Criterion']()
end

if opt.GPU ~= -1 then
    -- Convert model to CUDA
    print('==> Converting model to CUDA')
    model:cuda()
    criterion:cuda()

    cudnn.fastest = true
    cudnn.benchmark = true
end
```

A.2 HyperStackNet Testing Code

The following subsections are explanations of the various lua language files modified for the HyperStackNet training.

A.2.1 main.lua

The only line modified is line 81 where the function drawOutput is called from util.lua which draws and saves the skeleton of the image while also saving the heatmaps.

```
1  require 'paths'
2  paths.dofile('util.lua')
3  paths.dofile('img.lua')
4
5  ffi = require 'ffi'
6  --------------------------------------------------------------------------------
7  -- Initialization
8  --------------------------------------------------------------------------------
9
10 if arg[1] == 'demo' or arg[1] == 'predict-test' then
11     -- Test set annotations do not have ground truth part locations, but provide
12     -- information about the location and scale of people in each image.
13     a = loadAnnotations('annot_corrected')
14     local allIdxs = torch.range(1,a['index']:size(1))
15     test_idxs = torch.sort(allIdxs[a['istrain']:eq(0)])
16
17 elseif arg[1] == 'predict-valid' or arg[1] == 'eval' then
18     -- Validation set annotations on the other hand, provide part locations,
19     -- visibility information, normalization factors for final evaluation, etc.
20     a = loadAnnotations('valid')
21
22 else
23     print("Please use one of the following input arguments:")
24     print("    demo - Generate and display results on a few demo images")
25     print("    predict-valid - Generate predictions on the validation set (MPII images
26     print("    predict-test - Generate predictions on the test set")
27     print("    eval - Run basic evaluation on predictions from the validation set")
28     return
29 end
30
31 m = torch.load('hyperstacknet.t7')    -- Load pre-trained model
32
33 if arg[1] == 'demo' then
```

```
    idxs = test_idxs
else
    idxs = torch.range(1,a.nsamples)
end

if arg[1] == 'eval' then
    nsamples = 0
else
    nsamples = idxs:nElement()
    -- Displays a convenient progress bar
    xlua.progress(0,nsamples)
    preds = torch.Tensor(nsamples,18,64,64)
end

--------------------------------------------------------------------------------
-- Main loop
--------------------------------------------------------------------------------

for i = 1,nsamples do
    -- Set up input image
   local im = image.load('images/' .. ffi.string(a['imgname'][idxs[i]]:char():data()))
    local center = a['center'][idxs[i]]
    local scale = a['scale'][idxs[i]]
    scale = 1.8 * scale

    local inp = crop(im, center, scale, 0, 256)

    -- Get network output
    local out = m:forward(inp:view(1,3,256,256):cuda())
    out = applyFn(function (x) return x:clone() end, out)
            local flippedOut = m:forward(flip(inp:view(1,3,256,256):cuda()))
            flippedOut = applyFn(function (x) return flip(shuffleLR(x)) end, flippedOut
            out = applyFn(function (x,y) return x:add(y):div(2) end, out, flippedOut)
    cutorch.synchronize()
    local hm = out[#out][1]:float()
    hm[hm:lt(0)] = 0

    -- Get predictions (hm and img refer to the coordinate space)
```

```
72      local preds_hm, preds_img = getPreds(hm, center, scale)
73      preds[i]:copy(hm)
74
75      xlua.progress(i,nsamples)
76
77      -- Display the result
78      if arg[1] == 'demo' then
79          preds_hm:mul(4) -- Change to input scale
80
81          drawOutput(inp, hm, preds_hm[1],a,i)
82      end
83
84      collectgarbage()
85  end
86
87  -- Save predictions
88  if arg[1] == 'demo' then
89      local predFile = hdf5.open('preds/harpe_pred.h5', 'w')
90      predFile:write('preds', preds)
91      predFile:close()
92  elseif arg[1] == 'predict-test' then
93      local predFile = hdf5.open('preds/test.h5', 'w')
94      predFile:write('preds', preds)
95      predFile:close()
96  elseif arg[1] == 'demoo' then
97      w.window:close()
98  end
99
100 ----------------------------------------------------------------------------
101 -- Evaluation code
102 ----------------------------------------------------------------------------
103
104 if arg[1] == 'eval' then
105     -- Calculate distances given each set of predictions
106     local labels = {'valid-example','valid-ours'}
107     local dists = {}
108     for i = 1,#labels do
109         local predFile = hdf5.open('preds/' .. labels[i] .. '.h5','r')
```

```
110         local preds = predFile:read('preds'):all()
111         table.insert(dists,calcDists(preds, a.part, a.normalize))
112     end
113
114     require 'gnuplot'
115     gnuplot.raw('set bmargin 1')
116     gnuplot.raw('set lmargin 3.2')
117     gnuplot.raw('set rmargin 2')
118     gnuplot.raw('set multiplot layout 2,3 title "MPII Validation Set Performance (PCKh)
119     gnuplot.raw('set xtics font ",6"')
120     gnuplot.raw('set ytics font ",6"')
121     displayPCK(dists, {9,10}, labels, 'Head')
122     displayPCK(dists, {2,5}, labels, 'Knee')
123     displayPCK(dists, {1,6}, labels, 'Ankle')
124     gnuplot.raw('set tmargin 2.5')
125     gnuplot.raw('set bmargin 1.5')
126     displayPCK(dists, {13,14}, labels, 'Shoulder')
127     displayPCK(dists, {12,15}, labels, 'Elbow')
128     displayPCK(dists, {11,16}, labels, 'Wrist', true)
129     gnuplot.raw('unset multiplot')
130 end
```

A.2.2 img.lua

The only function added was compileImages18 in lines 199-230. This function modifies the original compileImages function by accommodating 18 heatmaps and joints as apposed to the original 16.

```
1 ----------------------------------------------------------------------
2 -- Coordinate transformation
3 ----------------------------------------------------------------------
4 function applyFn(fn, t, t2)
5     -- Apply an operation whether passed a table or tensor
6     local t_ = {}
7     if type(t) == "table" then
8         if t2 then
9             for i = 1,#t do t_[i] = applyFn(fn, t[i], t2[i]) end
```

```
        else
            for i = 1,#t do t_[i] = applyFn(fn, t[i]) end
        end
    else t_ = fn(t, t2) end
    return t_
end

function getTransform(center, scale, rot, res)
    local h = 200 * scale
    local t = torch.eye(3)

    -- Scaling
    t[1][1] = res / h
    t[2][2] = res / h

    -- Translation
    t[1][3] = res * (-center[1] / h + .5)
    t[2][3] = res * (-center[2] / h + .5)

    -- Rotation
    if rot ~= 0 then
        rot = -rot
        local r = torch.eye(3)
        local ang = rot * math.pi / 180
        local s = math.sin(ang)
        local c = math.cos(ang)
        r[1][1] = c
        r[1][2] = -s
        r[2][1] = s
        r[2][2] = c
        -- Need to make sure rotation is around center
        local t_ = torch.eye(3)
        t_[1][3] = -res/2
        t_[2][3] = -res/2
        local t_inv = torch.eye(3)
        t_inv[1][3] = res/2
        t_inv[2][3] = res/2
        t = t_inv * r * t_ * t
```

```
    end

    return t
end

function transform(pt, center, scale, rot, res, invert)
    -- For managing coordinate transformations between the original image space
    -- and the heatmap
    local pt_ = torch.ones(3)
    pt_[1] = pt[1]
    pt_[2] = pt[2]
    local t = getTransform(center, scale, rot, res)
    if invert then
        t = torch.inverse(t)
    end
    local new_point = (t*pt_):sub(1,2):int()
    return new_point
end

-------------------------------------------------------------------------------
-- Cropping
-------------------------------------------------------------------------------
function crop1(img, center, scale, rot, res)
    local ndim = img:nDimension()
    if ndim == 2 then img = img:view(1,img:size(1),img:size(2)) end
    local ht,wd = img:size(2), img:size(3)
    local tmpImg,newImg = img, torch.zeros(img:size(1), res, res)

    -- Modify crop approach depending on whether we zoom in/out
    -- This is for efficiency in extreme scaling cases
    local scaleFactor = (200 * scale) / res

    if scaleFactor < 2 then scaleFactor = 1
    else
        local newSize = math.floor(math.max(ht,wd) / scaleFactor)
        if newSize < 2 then
           -- Zoomed out so much that the image is now a single pixel or less
           if ndim == 2 then newImg = newImg:view(newImg:size(2),newImg:size(3)) end
```

```
            return newImg
        else
            tmpImg = image.scale(img,newSize)
            ht,wd = tmpImg:size(2),tmpImg:size(3)
        end
    end

    -- Calculate upper left and bottom right coordinates defining crop region
    local c,s = center:float()/scaleFactor, scale/scaleFactor
    local ul = transform({1,1}, c, s, 0, res, true)
    local br = transform({res+1,res+1}, c, s, 0, res, true)
    if scaleFactor >= 2 then br:add(-(br - ul - res)) end

    -- If the image is to be rotated, pad the cropped area
    local pad = math.ceil(torch.norm((ul - br):float())/2 - (br[1]-ul[1])/2)
    if rot ~= 0 then ul:add(-pad); br:add(pad) end

    -- Define the range of pixels to take from the old image
    local old_ = {1,-1,math.max(1, ul[2]), math.min(br[2], ht+1) - 1,
                       math.max(1, ul[1]), math.min(br[1], wd+1) - 1}
    -- And where to put them in the new image
    local new_ = {1,-1,math.max(1, -ul[2] + 2), math.min(br[2], ht+1) - ul[2],
                       math.max(1, -ul[1] + 2), math.min(br[1], wd+1) - ul[1]}

    -- Initialize new image and copy pixels over
    local newImg = torch.zeros(img:size(1), br[2] - ul[2], br[1] - ul[1])
    if not pcall(function() newImg:sub(unpack(new_)):copy(tmpImg:sub(unpack(old_))) end
       print("Error occurred during crop!")
    end

    if rot ~= 0 then
        -- Rotate the image and remove padded area
        newImg = image.rotate(newImg, rot * math.pi / 180, 'bilinear')
        newImg = newImg:sub(1,-1,pad+1,newImg:size(2)-pad,pad+1,newImg:size(3)-pad):clo
    end

    if scaleFactor < 2 then newImg = image.scale(newImg,res,res) end
    if ndim == 2 then newImg = newImg:view(newImg:size(2),newImg:size(3)) end
```

```
124     return newImg
125 end
126
127
128 function crop(img, center, scale, rot, res)
129     -- Crop function tailored to the needs of our system. Provide a center
130     -- and scale value and the image will be cropped and resized to the output
131     -- resolution determined by res. 'rot' will also rotate the image as needed.
132
133     local ul = transform({1,1}, center, scale, 0, res, true)
134     local br = transform({res,res}, center, scale, 0, res, true)
135
136     local pad = math.floor(torch.norm((ul - br):float())/2 - (br[1]-ul[1])/2)
137     if rot ~= 0 then
138         ul = ul - pad
139         br = br + pad
140     end
141
142     local newDim,newImg,ht,wd
143
144     if img:size():size() > 2 then
145         newDim = torch.IntTensor({img:size()[1], br[2] - ul[2], br[1] - ul[1]})
146         newImg = torch.zeros(newDim[1],newDim[2],newDim[3])
147         ht = img:size()[2]
148         wd = img:size()[3]
149     else
150         newDim = torch.IntTensor({br[2] - ul[2], br[1] - ul[1]})
151         newImg = torch.zeros(newDim[1],newDim[2])
152         ht = img:size()[1]
153         wd = img:size()[2]
154     end
155
156     local newX = torch.Tensor({math.max(1, -ul[1]+1), math.min(br[1], wd) - ul[1]})
157     local newY = torch.Tensor({math.max(1, -ul[2]+1), math.min(br[2], ht) - ul[2]})
158     local oldX = torch.Tensor({math.max(1, ul[1]+1), math.min(br[1], wd)})
159     local oldY = torch.Tensor({math.max(1, ul[2]+1), math.min(br[2], ht)})
160
161     if newDim:size()[1] > 2 then
```

```
162             newImg:sub(1,newDim[1],newY[1],newY[2],newX[1],newX[2]):copy(img:sub(1,newDim[1
163         else
164             newImg:sub(newY[1],newY[2],newX[1],newX[2]):copy(img:sub(oldY[1],oldY[2],oldX[1
165         end
166
167         if rot ~= 0 then
168             newImg = image.rotate(newImg, rot * math.pi / 180, 'bilinear')
169             if newDim:size()[1] > 2 then
170                 newImg = newImg:sub(1,newDim[1],pad,newDim[2]-pad,pad,newDim[3]-pad)
171             else
172                 newImg = newImg:sub(pad,newDim[1]-pad,pad,newDim[2]-pad)
173             end
174         end
175
176         newImg = image.scale(newImg,res,res)
177         return newImg
178     end
179
180     function twoPointCrop(img, s, pt1, pt2, pad, res)
181         local center = (pt1 + pt2) / 2
182         local scale = math.max(20*s,torch.norm(pt1 - pt2)) * .007
183         scale = scale * pad
184         local angle = math.atan2(pt2[2]-pt1[2],pt2[1]-pt1[1]) * 180 / math.pi - 90
185         return crop(img, center, scale, angle, res)
186     end
187
188     function compileImages(imgs, nrows, ncols, res)
189         -- Assumes the input images are all square/the same resolution
190         local totalImg = torch.zeros(3,nrows*res,ncols*res)
191         for i = 1,#imgs do
192             local r = torch.floor((i-1)/ncols) + 1
193             local c = ((i - 1) % ncols) + 1
194             totalImg:sub(1,3,(r-1)*res+1,r*res,(c-1)*res+1,c*res):copy(imgs[i])
195         end
196         return totalImg
197     end
198
199     function compileImages18(imgs, nrows, ncols, res)
```

```
200     -- Assumes the input images are all square/the same resolution
201     local totalImg = torch.zeros(3,nrows*res,ncols*res)
202 
203     for i = 1,#imgs do
204         local r = torch.ceil((i)/ncols)
205         if i%ncols == 0 then
206              c = 4
207         else
208              c = ((i) % ncols)
209         end
210 
211         totalImg:sub(1,3,(r-1)*res+1,r*res,(c-1)*res+1,c*res):copy(imgs[i])
212     end
213     return totalImg
214 end
215 
216 ----------------------------------------------------------------------------------
217 -- Non-maximum Suppression
218 ----------------------------------------------------------------------------------
219 
220 -- Set up max network for NMS
221 nms_window_size = 3
222 nms_pad = (nms_window_size - 1)/2
223 maxlayer = nn.Sequential()
224 if cudnn then
225     maxlayer:add(cudnn.SpatialMaxPooling(nms_window_size, nms_window_size,1,1, nms_pad,
226     maxlayer:cuda()
227 else
228     maxlayer:add(nn.SpatialMaxPooling(nms_window_size, nms_window_size,1,1, nms_pad,nms
229 end
230 maxlayer:evaluate()
231 
232 function local_maxes(hm, n, c, s, hm_idx)
233     hm = torch.Tensor(1,16,64,64):copy(hm):float()
234     if hm_idx then hm = hm:sub(1,-1,hm_idx,hm_idx) end
235     local hm_dim = hm:size()
236     local max_out
237     -- First do nms
```

```
238     if cudnn then
239         local hmCuda = torch.CudaTensor(1, hm_dim[2], hm_dim[3], hm_dim[4])
240         hmCuda:copy(hm)
241         max_out = maxlayer:forward(hmCuda)
242         cutorch.synchronize()
243     else
244         max_out = maxlayer:forward(hm)
245     end
246 
247     local nms = torch.cmul(hm, torch.eq(hm, max_out:float()):float())[1]
248     -- Loop through each heatmap retrieving top n locations, and their scores
249     local pred_coords = torch.Tensor(hm_dim[2], n, 2)
250     local pred_scores = torch.Tensor(hm_dim[2], n)
251     for i = 1, hm_dim[2] do
252         local nms_flat = nms[i]:view(nms[i]:nElement())
253         local vals,idxs = torch.sort(nms_flat,1,true)
254         for j = 1,n do
255             local pt = {idxs[j] % 64, torch.ceil(idxs[j] / 64) }
256             pred_coords[i][j] = transform(pt, c, s, 0, 64, true)
257             pred_scores[i][j] = vals[j]
258         end
259     end
260     return pred_coords, pred_scores
261 end
262 
263 ----------------------------------------------------------------------------
264 -- Drawing functions
265 ----------------------------------------------------------------------------
266 
267 function drawGaussian(img, pt, sigma)
268     -- Draw a 2D gaussian
269     -- Check that any part of the gaussian is in-bounds
270     local ul = {math.floor(pt[1] - 3 * sigma), math.floor(pt[2] - 3 * sigma)}
271     local br = {math.floor(pt[1] + 3 * sigma), math.floor(pt[2] + 3 * sigma)}
272     -- If not, return the image as is
273     if (ul[1] > img:size(2) or ul[2] > img:size(1) or br[1] < 1 or br[2] < 1) then retu
274     -- Generate gaussian
275     local size = 6 * sigma + 1
```

```
276     local g = image.gaussian(size) -- , 1 / size, 1)
277     -- Usable gaussian range
278     local g_x = {math.max(1, -ul[1]), math.min(br[1], img:size(2)) - math.max(1, ul[1])
279     local g_y = {math.max(1, -ul[2]), math.min(br[2], img:size(1)) - math.max(1, ul[2])
280     -- Image range
281     local img_x = {math.max(1, ul[1]), math.min(br[1], img:size(2))}
282     local img_y = {math.max(1, ul[2]), math.min(br[2], img:size(1))}
283     assert(g_x[1] > 0 and g_y[1] > 0)
284     img:sub(img_y[1], img_y[2], img_x[1], img_x[2]):add(g:sub(g_y[1], g_y[2], g_x[1], g
285     img[img:gt(1)] = 1
286     return img
287 end
288
289 function drawLine(img,pt1,pt2,width,color)
290     -- I'm sure there's a line drawing function somewhere in Torch,
291     -- but since I couldn't find it here's my basic implementation
292     local color = color or {1,1,1}
293     local m = torch.dist(pt1,pt2)
294     local dy = (pt2[2] - pt1[2])/m
295     local dx = (pt2[1] - pt1[1])/m
296     for j = 1,width do
297         local start_pt1 = torch.Tensor({pt1[1] + (-width/2 + j-1)*dy, pt1[2] - (-width/
298         start_pt1:ceil()
299         for i = 1,torch.ceil(m) do
300             local y_idx = torch.ceil(start_pt1[2]+dy*i)
301             local x_idx = torch.ceil(start_pt1[1]+dx*i)
302             if y_idx - 1 > 0 and x_idx -1 > 0 and y_idx < img:size(2) and x_idx < img:s
303                 img:sub(1,1,y_idx-1,y_idx,x_idx-1,x_idx):fill(color[1])
304                 img:sub(2,2,y_idx-1,y_idx,x_idx-1,x_idx):fill(color[2])
305                 img:sub(3,3,y_idx-1,y_idx,x_idx-1,x_idx):fill(color[3])
306             end
307         end
308     end
309     img[img:gt(1)] = 1
310
311     return img
312 end
313
```

```lua
function colorHM(x)
    -- Converts a one-channel grayscale image to a color heatmap image
    local function gauss(x,a,b,c)
        return torch.exp(-torch.pow(torch.add(x,-b),2):div(2*c*c)):mul(a)
    end
    local cl = torch.zeros(3,x:size(1),x:size(2))
    cl[1] = gauss(x,.5,.6,.2) + gauss(x,1,.8,.3)
    cl[2] = gauss(x,1,.5,.3)
    cl[3] = gauss(x,1,.2,.3)
    cl[cl:gt(1)] = 1
    return cl
end


--------------------------------------------------------------------------------
-- Flipping functions
--------------------------------------------------------------------------------

function shuffleLR(x)
    local dim
    if x:nDimension() == 4 then
        dim = 2
    else
        assert(x:nDimension() == 3)
        dim = 1
    end

    local matched_parts = {
        {1,6},   {2,5},   {3,4}
    }

    for i = 1,#matched_parts do
        local idx1, idx2 = unpack(matched_parts[i])
        local tmp = x:narrow(dim, idx1, 1):clone()
        x:narrow(dim, idx1, 1):copy(x:narrow(dim, idx2, 1))
        x:narrow(dim, idx2, 1):copy(tmp)
    end

```

```
352     return x
353 end
354
355 function flip(x)
356     require 'image'
357     local y = torch.FloatTensor(x:size())
358     for i = 1, x:size(1) do
359         image.hflip(y[i], x[i]:float())
360     end
361     return y:typeAs(x)
362 end
```

A.2.3 util.lua

Two functions were modified in this network. The first function, drawSkeleton in lines 55-93, draws skeleton features from the predictions overlaying on the original image. The second function, drawOutput in lines 95-110, saves the drawn skeleton while also creating and saving the heatmap images of the 18 joint locations.

```
1  require 'torch'
2  require 'xlua'
3  require 'nn'
4  require 'nnx'
5  require 'nngraph'
6  require 'image'
7  require 'hdf5'
8  require 'sys'
9
10 require 'cunn'
11 require 'cutorch'
12 require 'cudnn'
13
14 function loadAnnotations(set)
15     -- Load up a set of annotations for either: 'train', 'valid', or 'test'
16     -- There is no part information in 'test'
17
18     local a = hdf5.open('annot/' .. set .. '.h5')
```

```
    annot = {}

    -- -- Read in annotation information from hdf5 file
    local tags = {'part','center','scale','normalize','torsoangle','visible','imgname',
    for _,tag in ipairs(tags) do annot[tag] = a:read(tag):all() end
    annot.nsamples = annot.part:size()[1]
    a:close()
    return annot
end

function getPreds(hms, center, scale)
    if hms:size():size() == 3 then hms = hms:view(1, hms:size(1), hms:size(2), hms:size
    -- Get locations of maximum activations
    local max, idx = torch.max(hms:view(hms:size(1), hms:size(2), hms:size(3) * hms:siz
    local preds = torch.repeatTensor(idx, 1, 1, 2):float()
    preds[{{}, {}, 1}]:apply(function(x) return (x - 1) % hms:size(4) + 1 end)
    preds[{{}, {}, 2}]:add(-1):div(hms:size(3)):floor():add(1)
    local predMask = max:gt(0):repeatTensor(1, 1, 2):float()
    preds:add(-.5):cmul(predMask):add(1)

    -- Get transformed coordinates
    local preds_tf = torch.zeros(preds:size())
    for i = 1,hms:size(1) do        -- Number of samples
        for j = 1,hms:size(2) do    -- Number of output heatmaps for one sample
            preds_tf[i][j] = transform(preds[i][j],center,scale,0,hms:size(3),true)
        end
    end

    return preds, preds_tf
end

----------------------------------------------------------------------------
-- Functions for setting up the demo display
----------------------------------------------------------------------------

function drawSkeleton(input, hms, coords)

```

```
57      local im = input:clone()
58
59      local pairRef = {
60           {1,2},      {2,3},      {3,7},
61          {4,5},      {4,7},      {5,6},
62          {7,9},      {9,10},
63          {14,9},     {11,12},    {12,13},
64          {13,9},     {14,15},    {15,16},
65          {17,18}
66      }
67
68      local partNames = {'RAnk','RKne','RHip','LHip','LKne','LAnk',
69                         'Pelv','Thrx','Neck','Head',
70                         'RWri','RElb','RSho','LSho','LElb','LWri'}
71      local partColor = {1,1,1,2,2,2,0,0,0,0,3,3,3,4,4,4,5}
72
73      local actThresh = 0.000
74
75      -- Loop through adjacent joint pairings
76      for i = 1,#pairRef do
77          if hms[pairRef[i][1]]:mean() > actThresh and hms[pairRef[i][2]]:mean() > actThr
78              -- Set appropriate line color
79              local color
80              if partColor[pairRef[i][1]] == 1 then color = {0,.3,1}
81              elseif partColor[pairRef[i][1]] == 2 then color = {1,.3,0}
82              elseif partColor[pairRef[i][1]] == 3 then color = {0,0,1}
83              elseif partColor[pairRef[i][1]] == 4 then color = {1,0,0}
84              elseif partColor[pairRef[i][1]] == 5 then color = {1,0,0}
85              else color = {.7,0,.7} end
86
87              -- Draw line
88              im = drawLine(im, coords[pairRef[i][1]], coords[pairRef[i][2]], 4, color, 
89          end
90      end
91
92      return im
93  end
94
```

```lua
function drawOutput(input, hms, coords,a,i)

    local im = drawSkeleton(input, hms, coords)
    im = image.scale(im,756)
    image.save('preds/image_skeletons/' .. ffi.string(a['imgname'][idxs[i]]:char():data

    local colorHms = {}
    local inp64 = image.scale(input,64):mul(.3)
    for i = 1,18 do
        colorHms[i] = colorHM(hms[i])
        colorHms[i]:mul(.7):add(inp64)
    end
    local totalHm = compileImages18(colorHms, 5, 4, 64)
    totalHm = image.scale(totalHm,756)
    image.save('preds/heatmaps/' .. ffi.string(a['imgname'][idxs[i]]:char():data()), to
end

--------------------------------------------------------------------------------
-- Functions for evaluation
--------------------------------------------------------------------------------

function calcDists(preds, label, normalize)
    local dists = torch.Tensor(preds:size(2), preds:size(1))
    local diff = torch.Tensor(2)
    for i = 1,preds:size(1) do
        for j = 1,preds:size(2) do
            if label[i][j][1] > 1 and label[i][j][2] > 1 then
                dists[j][i] = torch.dist(label[i][j],preds[i][j])/normalize[i]
            else
                dists[j][i] = -1
            end
        end
    end
    return dists
end

function distAccuracy(dists, thr)
    -- Return percentage below threshold while ignoring values with a -1
```

```
    if not thr then thr = .5 end
    if torch.ne(dists,-1):sum() > 0 then
        return dists:le(thr):eq(dists:ne(-1)):sum() / dists:ne(-1):sum()
    else
        return -1
    end
end

function displayPCK(dists, part_idx, label, title, show_key)
    -- Generate standard PCK plot
    if not (type(part_idx) == 'table') then
        part_idx = {part_idx}
    end

    curve_res = 11
    num_curves = #dists
    local t = torch.linspace(0,.5,curve_res)
    local pdj_scores = torch.zeros(num_curves, curve_res)
    local plot_args = {}
    for curve = 1,num_curves do
        for i = 1,curve_res do
            t[i] = (i-1)*.05
            local acc = 0.0
            for j = 1,#part_idx do
                acc = acc + distAccuracy(dists[curve][part_idx[j]], t[i])
            end
            pdj_scores[curve][i] = acc / #part_idx
        end
        plot_args[curve] = {label[curve],t,pdj_scores[curve],'-'}
    end

    require 'gnuplot'
    gnuplot.raw('set title "' .. title .. '"')
    if not show_key then gnuplot.raw('unset key')
    else gnuplot.raw('set key font ",6" right bottom') end
    gnuplot.raw('set xrange [0:.5]')
    gnuplot.raw('set yrange [0:1]')
    gnuplot.plot(unpack(plot_args))
```

```
171 end
```

Appendix B

Code Base For Action Recognition

As previously mentioned in pose estimation, the initial pose estimation was provided by using the stacked hourglass network by Newell *et al.* [40]. To perform the action recognition, a transformer layer and feed forward layers were needed. The following code provides resources for the transformer layer as the stacked hourglass network can be found online whilst the feed forward layers were programmed using MATLAB Neural Network Toolbox. The transformation layers were programmed in Matlab and comprises of three files: FeatureCalculation(40D).m, RelativeAngl.m and VecAngl.m, where the last two files are helper functions for the main code (FeatureCalculation).

B.1 FeatureCalculation.m

This file is the main file that loads predicted results, calculates the relative angles between vectors of body parts, normalizes the angle with respect to the average head size and then concatenates the results with the joint locations to output a total of 40 features.

```
clc
clear all
% load all of the predictions
data1_ = h5read('~\Results\hockey-cross-turn.h5','/preds');
inds1 = xlsread('~\Results\Cross-Turn_GoodFrmNum.xlsx');
%--------------------------------------------------------------------------
data2_ = h5read('~\Results\hockey-firmFeet.h5','/preds');
```

```
inds2 = xlsread('~\Results\frimFeet_GoodFrmNum.xlsx');
%-------------------------------------------------------------------------------
data3_= h5read('~\Results\hockey-post-shot.h5','/preds');
inds3 = xlsread('~\Results\PostShoot_GoodFrmNum.xlsx');
%-------------------------------------------------------------------------------
data4_ = h5read('~\Results\hockey-pre-shot.h5','/preds');
inds4 = xlsread('~\Results\PreShoot_GoodFrmNum.xlsx');
%-------------------------------------------------------------------------------
data1 = data1_(: , : , inds1);
data2 = data2_(: , : , inds2);
data3 = data3_(: , : , inds3);
data4 = data4_(: , : , inds4);
%-------------------------------------------------------------------------------
[m1 ,n1 , p1] = size(data1);
[m2 ,n2 , p2] = size(data2);
[m3 ,n3 , p3] = size(data3);
[m4 ,n4 , p4] = size(data3);
vec = zeros(p1+p2+p3+p4 , 41);% 8 angles, (16 *2) joints coordinates, and 1 class
Ang = zeros(p1 , 8);
for i = 1 : p1
    joints = data1(: , : , i);
    %joint between 2 limbs has been considered the center, O, for calculating the ang
    %-----------------------------------
    % R-knee angle
    a = joints(: , 1);% R-ankle
    o = joints(: , 2);% R-knee
    b = joints(: , 3);% R-hip
    v1 = a - o;
    v2 = b - o;
    Ang(i,1)= RelativeAngl(v1 , v2);
    %-----------------------------------
    % L-knee angle
    a = joints(: , 4);% L-hip
    o = joints(: , 5);% L-knee
    b = joints(: , 6);% L-ankle
    v1 = a - o;
    v2 = b - o;
    Ang(i,2)=RelativeAngl(v1 , v2);
```

```
46      %-------------------------------------
47      % R-hip angle
48      a = joints(: , 2);% R-knee
49      o = joints(: , 3);% R-hip
50      b = joints(: , 7);% Pelvis
51      v1 = a - o;
52      v2 = b - o;
53      Ang(i,3)=RelativeAngl(v1 , v2);
54      %-------------------------------------
55      % L-hip angle
56      a = joints(: , 7);% Pelvis
57      o = joints(: , 4);% L-hip
58      b = joints(: , 5);% L-knee
59      v1 = a - o;
60      v2 = b - o;
61      Ang(i,4)= RelativeAngl(v1 , v2);
62      %-------------------------------------
63      % R-elbow angle
64      a = joints(: , 11);% R-wrist
65      o = joints(: , 12);% R-elbow
66      b = joints(: , 13);% R-shoulder
67      v1 = a - o;
68      v2 = b - o;
69      Ang(i,5)= RelativeAngl(v1 , v2);
70      %-------------------------------------
71      % L-elbow angle
72      a = joints(: , 14);% L-shoulder
73      o = joints(: , 15);% L-elbow
74      b = joints(: , 16);% L-wrist
75      v1 = a - o;
76      v2 = b - o;
77      Ang(i,6)= RelativeAngl(v1 , v2);
78      %-------------------------------------
79      % R-shoulder angle
80      a = joints(: , 12);% R-elbow
81      o = joints(: , 13);% R-shoulder
82      b = joints(: , 8);%  thorax
83      v1 = a - o;
```

```
84      v2 = b - o;
85      Ang(i,7)= RelativeAngl(v1 , v2);
86      %-------------------------------------
87      % L-shoulder angle
88      a = joints(: , 8);% thorax
89      o = joints(: , 14);% L-shoulder
90      b = joints(: , 15);% L-elbow
91      v1 = a - o;
92      v2 = b - o;
93      Ang(i,8)= RelativeAngl(v1 , v2);
94     %-------------------------------------
95  end
96
97  %%
98  vec(1:p1 , 1:8)= Ang;
99  %%
100 Ang = zeros(p2 , 8);
101 for i = 1 : p2
102     joints = data2(: , : , i);
103     %joint between 2 limbs has been considered the center, O, for calculating the ang
104     %-------------------------------------
105     % R-knee angle
106     a = joints(: , 1);% R-ankle
107     o = joints(: , 2);% R-knee
108     b = joints(: , 3);% R-hip
109     v1 = a - o;
110     v2 = b - o;
111     Ang(i,1)= RelativeAngl(v1 , v2);
112     %-------------------------------------
113     % L-knee angle
114     a = joints(: , 4);% L-hip
115     o = joints(: , 5);% L-knee
116     b = joints(: , 6);% L-ankle
117     v1 = a - o;
118     v2 = b - o;
119     Ang(i,2)=RelativeAngl(v1 , v2);
120     %-------------------------------------
121     % R-hip angle
```

```
122     a = joints(: , 2);% R-knee
123     o = joints(: , 3);% R-hip
124     b = joints(: , 7);% Pelvis
125     v1 = a - o;
126     v2 = b - o;
127     Ang(i,3)=RelativeAngl(v1 , v2);
128     %-------------------------------------
129     % L-hip angle
130     a = joints(: , 7);% Pelvis
131     o = joints(: , 4);% L-hip
132     b = joints(: , 5);% L-knee
133     v1 = a - o;
134     v2 = b - o;
135     Ang(i,4)= RelativeAngl(v1 , v2);
136     %-------------------------------------
137     % R-elbow angle
138     a = joints(: , 11);% R-wrist
139     o = joints(: , 12);% R-elbow
140     b = joints(: , 13);% R-shoulder
141     v1 = a - o;
142     v2 = b - o;
143     Ang(i,5)= RelativeAngl(v1 , v2);
144     %-------------------------------------
145     % L-elbow angle
146     a = joints(: , 14);% L-shoulder
147     o = joints(: , 15);% L-elbow
148     b = joints(: , 16);% L-wrist
149     v1 = a - o;
150     v2 = b - o;
151     Ang(i,6)= RelativeAngl(v1 , v2);
152     %-------------------------------------
153     % R-shoulder angle
154     a = joints(: , 12);% R-elbow
155     o = joints(: , 13);% R-shoulder
156     b = joints(: , 8);%  thorax
157     v1 = a - o;
158     v2 = b - o;
159     Ang(i,7)= RelativeAngl(v1 , v2);
```

```
160      %-------------------------------------
161      % L-shoulder angle
162      a = joints(: , 8);% thorax
163      o = joints(: , 14);% L-shoulder
164      b = joints(: , 15);% L-elbow
165      v1 = a - o;
166      v2 = b - o;
167      Ang(i,8)= RelativeAngl(v1 , v2);
168     %-------------------------------------
169  end
170  %%
171  vec(p1+1:p1+p2,1:8)= Ang;
172  %%
173  Ang = zeros(p3 , 8);
174  for i = 1 : p3
175      joints = data3(: , : , i);
176      %joint between 2 limbs has been considered the center, O, for calculating the ang
177      %-------------------------------------
178      % R-knee angle
179      a = joints(: , 1);% R-ankle
180      o = joints(: , 2);% R-knee
181      b = joints(: , 3);% R-hip
182      v1 = a - o;
183      v2 = b - o;
184      Ang(i,1)= RelativeAngl(v1 , v2);
185      %-------------------------------------
186      % L-knee angle
187      a = joints(: , 4);% L-hip
188      o = joints(: , 5);% L-knee
189      b = joints(: , 6);% L-ankle
190      v1 = a - o;
191      v2 = b - o;
192      Ang(i,2)=RelativeAngl(v1 , v2);
193      %-------------------------------------
194      % R-hip angle
195      a = joints(: , 2);% R-knee
196      o = joints(: , 3);% R-hip
197      b = joints(: , 7);% Pelvis
```

```
198     v1 = a - o;
199     v2 = b - o;
200     Ang(i,3)=RelativeAngl(v1 , v2);
201     %-------------------------------------
202     % L-hip angle
203     a = joints(: , 7);% Pelvis
204     o = joints(: , 4);% L-hip
205     b = joints(: , 5);% L-knee
206     v1 = a - o;
207     v2 = b - o;
208     Ang(i,4)= RelativeAngl(v1 , v2);
209     %-------------------------------------
210     % R-elbow angle
211     a = joints(: , 11);% R-wrist
212     o = joints(: , 12);% R-elbow
213     b = joints(: , 13);% R-shoulder
214     v1 = a - o;
215     v2 = b - o;
216     Ang(i,5)= RelativeAngl(v1 , v2);
217     %-------------------------------------
218     % L-elbow angle
219     a = joints(: , 14);% L-shoulder
220     o = joints(: , 15);% L-elbow
221     b = joints(: , 16);% L-wrist
222     v1 = a - o;
223     v2 = b - o;
224     Ang(i,6)= RelativeAngl(v1 , v2);
225     %-------------------------------------
226     % R-shoulder angle
227     a = joints(: , 12);% R-elbow
228     o = joints(: , 13);% R-shoulder
229     b = joints(: , 8);%  thorax
230     v1 = a - o;
231     v2 = b - o;
232     Ang(i,7)= RelativeAngl(v1 , v2);
233     %-------------------------------------
234     % L-shoulder angle
235     a = joints(: , 8);% thorax
```

```
    o = joints(: , 14);% L-shoulder
    b = joints(: , 15);% L-elbow
    v1 = a - o;
    v2 = b - o;
    Ang(i,8)= RelativeAngl(v1 , v2);
   %-------------------------------------
end
%%
vec(p1+p2+1:p1+p2+p3,1:8)= Ang;
%%
%%
Ang = zeros(p4 , 8);
for i = 1 : p4
    joints = data4(: , : , i);
    %joint between 2 limbs has been considered the center, O, for calculating the ang
    %-------------------------------------
    % R-knee angle
    a = joints(: , 1);% R-ankle
    o = joints(: , 2);% R-knee
    b = joints(: , 3);% R-hip
    v1 = a - o;
    v2 = b - o;
    Ang(i,1)= RelativeAngl(v1 , v2);
    %-------------------------------------
    % L-knee angle
    a = joints(: , 4);% L-hip
    o = joints(: , 5);% L-knee
    b = joints(: , 6);% L-ankle
    v1 = a - o;
    v2 = b - o;
    Ang(i,2)=RelativeAngl(v1 , v2);
    %-------------------------------------
    % R-hip angle
    a = joints(: , 2);% R-knee
    o = joints(: , 3);% R-hip
    b = joints(: , 7);% Pelvis
    v1 = a - o;
    v2 = b - o;
```

```
274     Ang(i,3)=RelativeAngl(v1 , v2);
275     %-------------------------------------
276     % L-hip angle
277     a = joints(: , 7);% Pelvis
278     o = joints(: , 4);% L-hip
279     b = joints(: , 5);% L-knee
280     v1 = a - o;
281     v2 = b - o;
282     Ang(i,4)= RelativeAngl(v1 , v2);
283     %-------------------------------------
284     % R-elbow angle
285     a = joints(: , 11);% R-wrist
286     o = joints(: , 12);% R-elbow
287     b = joints(: , 13);% R-shoulder
288     v1 = a - o;
289     v2 = b - o;
290     Ang(i,5)= RelativeAngl(v1 , v2);
291     %-------------------------------------
292     % L-elbow angle
293     a = joints(: , 14);% L-shoulder
294     o = joints(: , 15);% L-elbow
295     b = joints(: , 16);% L-wrist
296     v1 = a - o;
297     v2 = b - o;
298     Ang(i,6)= RelativeAngl(v1 , v2);
299     %-------------------------------------
300     % R-shoulder angle
301     a = joints(: , 12);% R-elbow
302     o = joints(: , 13);% R-shoulder
303     b = joints(: , 8);%  thorax
304     v1 = a - o;
305     v2 = b - o;
306     Ang(i,7)= RelativeAngl(v1 , v2);
307     %-------------------------------------
308     % L-shoulder angle
309     a = joints(: , 8);% thorax
310     o = joints(: , 14);% L-shoulder
311     b = joints(: , 15);% L-elbow
```

```
312     v1 = a - o;
313     v2 = b - o;
314     Ang(i,8)= RelativeAngl(v1 , v2);
315    %-----------------------------------
316 end
317 %%
318 vec(p1+p2+p3+1:p1+p2+p3+p4,1:8)= Ang;
319 vec( isnan(vec)) = 0;% for nan angles put 0
320 %%
321 %-------------------------------------------------------------------------------------
322 for i = 1 : p1
323     head = data1(: , 9:10 , i);
324     hsz1(i) = sqrt(sum( (head(:, 1)- head(: , 2)).^2) );
325  end
326 %-----------------------------------------------------
327 for i = 1 : p2
328     head = data2(: , 9:10 , i);
329     hsz2(i) = sqrt(sum( (head(:, 1)- head(: , 2)).^2) );
330 end
331 %-----------------------------------------------------
332 for i = 1 : p3
333     head = data3(: , 9:10 , i);
334     hsz3(i) = sqrt(sum( (head(:, 1)- head(: , 2)).^2) );
335 end
336 %-----------------------------------------------------
337 for i = 1 : p4
338     head = data4(: , 9:10 , i);
339     hsz4(i) = sqrt(sum( (head(:, 1)- head(: , 2)).^2) );
340 end
341 %-----------------------------------------------------
342 Mhsz = mean([hsz1 , hsz2 ,hsz3, hsz4]);
343 scale1 = Mhsz ./ hsz1 ;
344 scale2 = Mhsz ./ hsz2 ;
345 scale3 = Mhsz ./ hsz3 ;
346 scale4 = Mhsz ./ hsz4 ;
347 %-----------------------------------------------------
348 % do scaling and find the new position of origin
349 for i = 1 : p1
```

```
350     data1(: , : , i) = data1(: , : , i) * scale1(i);
351     bd = data1(: , 7:8 , i);
352     O1 = bd(: , 1) + (bd(:, 2) -  bd(: , 1))/2;
353     data1 ( : , : , i) = data1 ( : , : , i) - repmat(O1 , [1,16]);
354     tmp = data1( : ,: , i);
355     vec(i , 9 : 40)= tmp(:);
356     vec(i , 41)= 1;
357 end
358 for i = 1 : p2
359     data2(: ,: ,i) = data2(: , : , i) * scale2(i);
360     bd = data2(: , 7:8 , i);
361     O2 = bd(: , 1) + (bd(:, 2) -  bd(: , 1))/2;
362     data2 ( : , : , i) = data2 ( : , : , i) - repmat(O2 , [1,16]);
363     tmp = data2( : ,: , i);
364     vec(p1 + i , 9 : 40)= tmp(:);
365     vec(p1 + i , 41)= 2;
366 end
367 for i = 1 : p3
368     data3(: ,: ,i) = data3(: , : , i) * scale3(i);
369     bd = data3(: , 7:8 , i);
370     O3 = bd(: , 1) +(bd(:, 2) -  bd(: , 1))/2;
371     data3 ( : , : , i) = data3 ( : , : , i) - repmat(O3 , [1,16]);
372     tmp = data3( : ,: , i);
373     vec(p1 + p2 + i , 9 : 40)= tmp(:);
374     vec(p1 + p2 + i , 41)= 3;
375 end
376 for i = 1 : p4
377     data4(: ,: ,i) = data4(: , : , i) * scale4(i);
378     bd = data4(: , 7:8 , i);
379     O4 = bd(: , 1) +(bd(:, 2) -  bd(: , 1))/2;
380     data4 ( : , : , i) = data4 ( : , : , i) - repmat(O4 , [1,16]);
381     tmp = data4( : ,: , i);
382     vec(p1 + p2 + p3 + i , 9 : 40)= tmp(:);
383     vec(p1 + p2 + p3 + i , 41)= 4;
384 end
385 %-------------------------------------------------
386
387 save('~\Results\vector.mat','vec')
```

B.2 RelativeAngle.m

This file contains the RelativeAngl function that calculates the relative angle of two vectors, which, in this case are body/stick parts. This calculation is done within the image frame.

```
function theta = RelativeAngl(v1 , v2)

theta1 = VecAngl(v1);
theta2 = VecAngl(v2);
theta = abs(theta1 - theta2);
if theta > 180
    theta = theta - 360;
end

end
```

B.3 VecAngle.m

This file contains the VecAngl function that calculates the angle between two joints relative to the first joint. This calculation is done within the image frame.

```
function alpha = VecAngl(v)
%quarter identification
if  ( v(1)> 0 && v(2)> 0) %qI
    alpha = (180/pi) * atan(v(2)/v(1));


elseif ( v(1)< 0 && v(2)> 0)%qII
    alpha = 180 + (180/pi) * atan(v(2)/v(1));


elseif ( v(1)< 0 && v(2)< 0)%qIII
    alpha = 180 + (180/pi) * atan(v(2)/v(1));


else%( v(1)> 0 && v(2)< 0)%qIV
```

```
16      alpha = 360 + (180/pi) * atan(v(2)/v(1));
17  end
18
19  end
```

References

[1] M. Andriluka, L. Pishchulin, P. Gehler, and B. Schiele. Human pose estimation: New benchmark and state of the art analysis. In *Proceedings IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, Columbus, USA, June 2014.

[2] L. Bourdev and J. Malik. Poselets: Body part detectors trained using 3d human pose annotations. In *2009 IEEE 12th International Conference on Computer Vision*, pages 1365–1372, Sept 2009.

[3] A. Bulat and G. Tzimiropoulos. Human pose estimation via convolutional part heatmap regression. In *14th European Conference on Computer Vision*, 2016.

[4] Y. Cai. Robust visual tracking for multiple targets. In *2006 Proceedings of the European Conference on Computer Vision (ECCV)*, pages 107–118, Graz,Austria, 2006.

[5] Z. Cao, T. Simon, S. E. Wei, and Y. Sheikh. Realtime multi-person 2d pose estimation using part affinity fields. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1302–1310, July 2017.

[6] J. Carreira, P. Agrawal, K. Fragkiadaki, and J. Malik. Human pose estimation with iterative error feedback. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[7] X. Chen and A. Yuille. Articulated pose estimation by a graphical model with image dependent pairwise relations. In *Advances in Neural Information Processing Systems (NIPS)*, 2014.

[8] X. Chen and A. L. Yuille. Parsing occluded people by flexible compositions. In *CVPR*, pages 3945–3954. IEEE Computer Society, 2015.

[9] Y. Chen, C. Shen, X. Wei, L. Liu, and J. Yang. Adversarial PoseNet: a structure-aware convolutional network for human pose estimation. In *International Conference on Computer Vision (ICCV'17)*, 2017.

[10] L. Cheng, Y. Guan, K. Zhu, and Y. Li. Recognition of human activities using machine learning methods with wearable sensors. In *2017 IEEE 7th Annual Computing and Communication Workshop and Conference (CCWC)*, pages 1–7, Las Vegas, USA, Jan 2017.

[11] A. Cherian, J. Mairal, K. Alahari, and C. Schmid. Mixing body-part sequences for human pose estimation. In *2014 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2361–2368, Columbus, USA, June 2014.

[12] G. Chéron, I. Laptev, and C. Schmid. P-cnn: Pose-based cnn features for action recognition. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 3218–3226, Santiago, Chile, 2015.

[13] C. Chou, J. Chien, and H. Chen. Fusing spatiotemporal features and joints for 3d action recognition. In *2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, Honolulu, USA, July 2017.

[14] X. Chu, W. Yang, W. Ouyang, C. Ma, A. L. Yuille, and X. Wang. Multi-context attention for human pose estimation. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5669–5678, July 2017.

[15] D. Conigliaro, P. Rota, F. Setti, C. Bassetti, N. Conci, N. Sebe, and M. Cristani. The S-HOCK dataset: Analyzing crowds at the stadium. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2039–2047, Boston, USA, June 2015.

[16] C. Desai and D. Ramanan. Detecting actions, poses, and objects with relational phraselets. In *Proceedings of the 12th European Conference on Computer Vision (ECCV) - Volume Part IV*, ECCV'12, Florence, Italy, 2012.

[17] W. Du, Y. Wang, and Y. Qiao. Rpan: An end-to-end recurrent pose-attention network for action recognition in videos. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 3745–3754, Oct 2017.

[18] N. Elliott, S. Choppin, S. Goodwill, T. Senior, J. Hart, and T. Allen. Single view silhouette fitting techniques for estimating tennis racket position. *Sports Engineering*, Jul 2017.

[19] N. Elliott, S. Choppin, S. R. Goodwill, and T. Allen. Markerless tracking of tennis racket motion using a camera. volume 72, pages 344 – 349, 2014. The Engineering of Sport 10.

[20] J. Fan, Z. Zha, and X. Tian. Action recognition with novel high-level pose features. In *2016 IEEE International Conference on Multimedia Expo Workshops (ICMEW)*, pages 1–6, July 2016.

[21] X. Fan, K. Zheng, Y. Lin, and S. Wang. Combining local appearance and holistic view: Dual-source deep neural networks for human pose estimation. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, April 2015.

[22] M. Fani, H. Neher, D. A. Clausi, A. Wong, and J. Zelek. Hockey action recognition via integrated stacked hourglass network. In *2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 85–93, Honolulu, USA, July 2017.

[23] P. Felzenszwalb, D. McAllester, and D. Ramanan. A discriminatively trained, multiscale, deformable part model. In *2008 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8, June 2008.

[24] V. Ferrari, M. Marin-Jimenez, and A. Zisserman. Progressive search space reduction for human pose estimation. In *2008 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8, June 2008.

[25] A. Gupta, J. J. Little, and R. J. Woodham. Using line and ellipse features for rectification of broadcast hockey video. In *The 14th Canadian Conference on Computer and Robot Vision (CRV'15)*, pages 32–39, Halifax, Canada, 2011. IEEE Computer Society.

[26] E. Insafutdinov, M. Andriluka, L. Pishchulin, S. Tang, E. Levinkov, B. Andres, and B. Schiele. Arttrack: Articulated multi-person tracking in the wild. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1293–1301, July 2017.

[27] A. Jain, J. Tompson, M. Andriluka, G. W. Taylor, and C. Bregler. Learning Human Pose Estimation Features with Convolutional Networks. In *IEEE Conference in Computer Vision and Pattern Recognition (CVPR)*, December 2013.

[28] A. Jain, J. Tompson, Y. LeCun, and C. Bregler. Modeep: A deep learning framework using motion features for human pose estimation. In *Computer Vision - ACCV 2014*

- *12th Asian Conference on Computer Vision, Revised Selected Papers*, volume 9004 of *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, pages 302–315. Springer Verlag, 2015.

[29] H. Jhuang, J. Gall, S. Zuffi, C. Schmid, and M. J. Black. Towards understanding action recognition. In *2013 IEEE International Conference on Computer Vision (ICCV)*, pages 3192–3199, Sydney, Australia, Dec 2013.

[30] S. Johnson and M. Everingham. Learning effective human pose estimation from inaccurate annotation. In *CVPR 2011*, pages 1465–1472, June 2011.

[31] L. Ladick, P. H. S. Torr, and A. Zisserman. Human pose estimation using a joint pixel-wise and part-wise formulation. In *2013 IEEE Conference on Computer Vision and Pattern Recognition*, pages 3578–3585, June 2013.

[32] C. Li, Y. Hou, P. Wang, and W. Li. Joint distance maps based action recognition with convolutional neural networks. *IEEE Signal Processing Letters*, 24(5):624–628, May 2017.

[33] F. Li and R. J. Woodham. Video analysis of hockey play in selected game situations. *Image and Vision Computing*, 27(12):45 – 58, 2009.

[34] J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3431–3440, June 2015.

[35] W. Lu and J. J. Little. Simultaneous tracking and action recognition using the PCA-HOG descriptor. In *The 3rd Canadian Conference on Computer and Robot Vision (CRV'06)*, pages 6–6, Quebec, Canada, June 2006.

[36] W. Lu and J. J. Little. Tracking and recognizing actions at a distance. In *Proceedings of the ECCV Workshop on Computer Vision Based Analysis in Sport Environments (CVBASE '06)*, Graz, Austria, May 2006.

[37] W. Lu, K. Okuma, and J. J. Little. Tracking and recognizing actions of multiple hockey players using the boosted particle filter. *Image and Vision Computing*, 27(1–2):189–205, 2009.

[38] H. Neher, M. Fani, D. A. Clausi, A. Wong, and J. Zelek. Pose estimation of players in hockey videos using convolutional neural networks. In *2017 Ottawa Hockey Analytics Conference (OTTHAC)*, Ottawa, Canada, 2017. Best Student Paper.

[39] H. Neher, K. Vats, A. Wong, and D. A. Clausi. Hyperstacknet: A hyper stacked hourglass deep convolutional neural network architecture for joint player and stick pose estimation in hockey. In *15th Conference on Computer and Robot Vision (CRV)*, York, Canada, 2018.

[40] A. Newell, K. Yang, and J. Deng. Stacked hourglass networks for human pose estimation. In Bastian Leibe, Jiri Matas, Nicu Sebe, and Max Welling, editors, *2016 Proceedings of the European Conference on Computer Vision (ECCV)*, pages 483–499, Amsterdam, Netherlands, October 2016. Springer International Publishing.

[41] B. X. Nie, C. Xiong, and S. C. Zhu. Joint action recognition and pose estimation from video. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1293–1301, Boston, USA, June 2015.

[42] G. Ning, Z. Zhang, and Z. He. Knowledge-guided deep fractal neural networks for human pose estimation. *IEEE Transactions on Multimedia*, 20(5):1246–1259, May 2018.

[43] K. Okuma, D. G. Lowe, and J. J. Little. Self-learning for player localization in sports video. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2013.

[44] K. Okuma, A. Taleghani, N. De Freitas, J. J. Little, and D. G. Lowe. A boosted particle filter: Multitarget detection and tracking. In *2004 Proceedings of the European Conference on Computer Vision (ECCV)*, pages 28–39, Prague, Czech Republic, 2004.

[45] G. L. Oliveira, A. Valada, C. Bollen, W. Burgard, and T. Brox. Deep learning for human part discovery in images. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1634–1641, May 2016.

[46] K. Papadopoulos, M. Antunes, D. Aouada, and B. Ottersten. Enhanced trajectory-based action recognition using human pose. In *2017 IEEE International Conference on Image Processing (ICIP)*, pages 1807–1811, Sept 2017.

[47] T. Pfister, J. Charles, and A. Zisserman. Flowing convnets for human pose estimation in videos. In *International Conference on Computer Vision (ICCV)*, 2015.

[48] L. Pishchulin, M. Andriluka, P. Gehler, and B. Schiele. Poselet conditioned pictorial structures. In *2013 IEEE Conference on Computer Vision and Pattern Recognition*, pages 588–595, June 2013.

[49] L. Pishchulin, M. Andriluka, P. Gehler, and B. Schiele. Strong appearance and expressive spatial models for human pose estimation. In *2013 IEEE International Conference on Computer Vision*, pages 3487–3494, Dec 2013.

[50] L. Pishchulin, M. Andriluka, and B. Schiele. Fine-grained activity recognition with holistic and pose based features. In *2014 German Conference on Pattern Recognition (GCPR)*, pages 678–689, Münster, Germany, 2014. Springer International Publishing.

[51] L. Pishchulin, E. Insafutdinov, S. Tang, B. Andres, M. Andriluka, P. Gehler, and B. Schiele. Deepcut: Joint subset partition and labeling for multi person pose estimation. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4929–4937, June 2016.

[52] D. Ramanan. Learning to parse images of articulated bodies. In *Proceedings of the 19th International Conference on Neural Information Processing Systems*, NIPS'06, pages 1129–1136, Cambridge, MA, USA, 2006. MIT Press.

[53] B. Sapp and B. Taskar. Modec: Multimodal decomposable models for human pose estimation. In *2013 IEEE Conference on Computer Vision and Pattern Recognition*, pages 3674–3681, June 2013.

[54] J. Shotton, A. Fitzgibbon, M. Cook, T. Sharp, M. Finocchio, R. Moore, A. Kipman, and A. Blake. Real-time human pose recognition in parts from single depth images. In *CVPR 2011*, pages 1297–1304, June 2011.

[55] J. Tompson, A. Jain, Y. LeCun, and C. Bregler. Joint training of a convolutional network and a graphical model for human pose estimation. In *IEEE Conference in Computer Vision and Pattern Recognition (CVPR)*, 2014.

[56] M. R. Tora, J. Chen, and J. J. Little. Classification of puck possession events in ice hockey. In *2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 147–154, July 2017.

[57] A. Toshev and C. Szegedy. Deeppose: Human pose estimation via deep neural networks. In *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1653–1660, June 2014.

[58] M. Garbade U. Iqbal and J. Gall. Pose for action - action for pose. In *Proceedings of the 12th IEEE International Conference on Automatic Face and Gesture Recognition (FG)*, Washington, DC, USA, May 2017.

[59] J. Wang, Z. Liu, Y. Wu, and J. Yuan. Mining actionlet ensemble for action recognition with depth cameras. In *2012 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1290–1297, Providence, USA, June 2012.

[60] J. Wang, X. Nie, Y. Xia, Y. Wu, and S. C. Zhu. Cross-view action modeling, learning, and recognition. In *2014 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2649–2656, June 2014.

[61] S. E. Wei, V. Ramakrishna, T. Kanade, and Y. Sheikh. Convolutional pose machines. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4724–4732, June 2016.

[62] J. Yang, Q. Liu, and K. Zhang. Stacked hourglass network for robust facial landmark localisation. In *2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 2025–2033, July 2017.

[63] W. Yang, S. Li, W. Ouyang, H. Li, and X. Wang. Learning feature pyramids for human pose estimation. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 1290–1299, Oct 2017.

[64] W. Yang, W. Ouyang S. Li, H. Li, and X. Wang. Learning feature pyramids for human pose estimation. In *Proceedings Ninth IEEE International Conference on Computer Vision*, Oct 2017.

[65] W. Yang, Y. Wang, and G. Mori. Recognizing human actions from still images with latent poses. In *2010 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2030–2037, San Francisco, USA, June 2010.

[66] X. Yang and Y. Tian. Super normal vector for human activity recognition with depth cameras. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(5):1028–1039, May 2017.

[67] Y. Yang and D. Ramanan. Articulated pose estimation with flexible mixtures-of-parts. In *2011 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1385–1392, June 2011.

[68] B. Yao and L. Fei-Fei. Recognizing human-object interactions in still images by modeling the mutual context of objects and human poses. volume 34, pages 1691–1703, Sept 2012.

[69] Z. Zhang, J. Hao, Y. Wang, and Y. Zhao. Enhanced human parsing with multiple feature fusion and augmented pose model. In *2014 22nd International Conference on Pattern Recognition*, pages 369–374, Aug 2014.

[70] Y. Zhou, J. Deng, and S. Zafeiriou. Improve accurate pose alignment and action localization by dense pose estimation. In *2018 13th IEEE International Conference on Automatic Face Gesture Recognition (FG 2018)*, pages 480–484, May 2018.