Identifying Sports Players in Broadcast Videos using Recurrent and Convolutional Neural Networks

Alvin Chan

Master of Engineering

Electrical and Computer Engineering

McGill University

Montreal,Quebec

August 2018

A thesis submitted to McGill University in partial fulfillment of the requirements of the degree of Master of Engineering

DEDICATION

This thesis is dedicated to my supportive family and friends.

ACKNOWLEDGEMENTS

I would like to first thank my supervisor, Prof. Martin Levine for his encouragement, patience, and support throughout the duration of this research project. I am grateful for his valuable teachings and greatly appreciate the guidance he has provided me to come this far.

I would also like to express my gratitude to Dr. Mehrsan Javan and the team at SPORTLOGiQ for the helpful discussions and insightful suggestions throughout this work. I am thankful to have their support and their work is an inspiration to my own.

This work was supported in part by SPORTLOGiQ and the MITACS Accelerate program. The GPUs, data, and other resources used in this research project were provided by SPORTLOGiQ.

All material related to SPORTLOGiQ intellectual property is presented in this thesis with permission.

ABSTRACT

In this work, we present a deep recurrent convolutional neural network approach to solve the problem of hockey player identification in NHL broadcast videos. Player identification is a difficult computer vision problem mainly because of the players' similar appearance, occlusion, and blurry facial and physical features. However, due to the nature of broadcast videos, we can observe players over time by processing variable length image sequences of players (aka 'tracklets'). Since the jersey number is the most distinguishable feature of a player, we propose an end-to-end trainable ResNet+LSTM network to learn features of jersey numbers and allow information from previous frames to influence the predictions for future time-steps. This network is based on a so-called residual network (ResNet) and a long short-term memory (LSTM) layer to discover spatio-temporal information and learn long-term dependencies. To train our model, we create a new NHL Hockey Player Tracklet (NHL-HPT) dataset that contains sequences of player bounding boxes. Additionally, we employ a secondary 1-dimensional CNN classifier as a late score-level fusion method to classify the output of the ResNet+LSTM network. This achieves an overall player identification accuracy score up to 87% on the new NHL-HPT dataset.

ABRÉGÉ

Dans ce travail, nous présentons une approche basée sur un réseau neuronal convolutionnels récurrents profonds pour résoudre le problème de l'identification des joueurs de hockey dans les vidéos diffusées de la LNH. L'identification du joueur est un problème de vision par ordinateur difficile, principalement de l'apparence semblable, de l'occlusion et des traits physiques et faciaux flous des joueurs. À cause de la nature de la vidéo, nous pouvons observer les joueurs au fil du temps par le traitement des séquences d'images de longueur variable de joueurs (alias «tracklets»). Puisque le numéro de maillot est la caractéristique la plus distinctive d'un joueur, nous proposons un réseau ResNet+LSTM de bout-en-bout entrainable, pour apprendre les caractéristiques des numéros de maillots et permettre aux informations des photogrammes précédentes de influencer les prédictions pour les prochaines étapes de temps. Ce réseau combine une base de réseau résiduel (ResNet) et une couche d'un réseau récurrent à mémoire court et long terme (LSTM) pour découvrir des informations spatio-temporelles et apprendre des dépendances à long terme. Pour s'entraîner notre modèle, nous créons un dataset nouveau du Tracklet des Joueurs de Hockey LNH (TJH-LNH) qui contient des séquences de boîtes de délimitation de joueurs. De plus, nous utilisons un classificateur CNN unidimensionnel secondaire comme méthode de fusion au niveau de score tardif pour classer la sortie du réseau ResNet+LSTM. Ca va obtenir un taux d'exactitude global d'identification des joueurs allant jusqu'à 87% sur le dataset nouveau du TJH-LNH.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1
Introduction

Ice hockey is a widely popular sport in North America that attracts millions of viewers during the National Hockey League (NHL) Stanley Cup playoffs. Recently, sports analytics has taken the hockey industry by storm and has become a major focus for the NHL. The data and results gathered from sports analytics allows greater insight into the players' abilities and game plays. Not only can sports analytics help coaches make better informed decisions to make the best use of a player's potential, this information can also be incorporated into TV broadcast videos to enhance the viewer's experience. More interesting storylines can be told and supported with actual data rather than relying on a coach's or analyst's opinion.

Currently, hockey games can be annotated manually by humans to track and identify players and record all events that occur throughout the game, but it is a laborious and time-intensive process. For sports analytics to become a norm in the industry, a fully automatic player detection, tracking, and identification system is required. To completely annotate a game, an action recognition system will also be needed to associate actions with the identified players.

In this work, we aim to contribute to building a full annotating system by focusing on the task of automatic hockey player identification in NHL broadcast videos using computer vision and deep learning methods. Although our focus is on hockey players, this task can also be applied to other team sports, like soccer and basketball. Nonetheless, identifying players in any sports broadcast video is a challenging problem. NHL broadcast videos are often recorded from the main camera, which is at a significant distance away from the hockey rink to provide viewers an encompassing view of the players on the rink. From a distance away, sport players of the same team have very similar appearance due to the near identical team jerseys and their facial and physical body features are almost indistinguishable from each other. Also, players are constantly moving fast on the ice in an unpredictable pattern and the camera itself is also in motion. With all the motion in the video, players can become blurry and often occlude each other from the view of the main camera. However, the jersey number on the back of every player is the most observable feature that can be used to identify each player. Although the jersey number is not always visible in the camera view, we can take advantage of the moments when the jersey number is fully visible.

This project was supported by SPORTLOGiQ [1] to explore different player identification methods for sports player tracking. The player tracking data provides us with short segments of the players' trajectories during a game. We refer to these as player tracklets. These player tracklets contain player location information, which we can use to extract bounding boxes around players from video frames to create image sequences of players. We manually extracted and annotated a set of image sequences of players to create a new NHL Hockey Player Tracklets (NHL-HPT) dataset to test our models.

[1] http://sportlogiq.com/en/

Given an image sequence of a player, our objective is identity the player within the image sequence by his jersey number and assign a single label to the sequence. Since we are considering multiple images together, we have a higher chance of observing the jersey number compared to analyzing each frame individually. Encouraged by the state-of-the-art results in street number recognition [5], we apply convolutional neural networks (CNN) to perform jersey number recognition. However, since the data we are working with are not single image inputs, we can approach our player identification problem as a video classification problem, which involves sequential inputs of multiple video frames.

Our approach is based on a recurrent convolutional neural network that is end-to-end trainable in a supervised learning setting. The network processes frames at each time-step using a CNN to generate a feature vector representation for each frame. In order to allow information to flow across time-steps and make use of information from features of previous frames, we use long short-term memory [6] (LSTM) units in the recurrent layer of the network. With this network, we can identify players solely based on their visual features without relying on any priors, such as game or player context.

The contributions of this work is two-fold. First, as far we know, our work is the first to perform player identification in NHL broadcast hockey videos, as no other work in the literature has involved hockey players. Second, we demonstrate that employing a recurrent convolutional neural network in our approach can produce excellent results in player identification.

The rest of this paper is structured as follows. In the next chapter, we will present the technical background on the deep neural networks used in this project and a literature review on the related work. In Chapter 3, we will discuss the datasets we used and the methodology behind our proposed approach. Chapter 4 will present our experimental implementations, results, and discussions. Finally, Chapter 5 will conclude this paper and provide insight on possible future work.

CHAPTER 2
Related Work

In this chapter, we will present the background of the models used in our work and a literature review on sports player identification and video classification. We will first provide a technical overview of the details of convolutional neural networks (CNNs) and the different architectures that were studied in this work. We will also examine the mechanics of recurrent neural networks (RNNs), including one of the more widely used variations, the long short-term memory (LSTM). We hope to provide the reader with a deeper understanding of neural networks before we begin to discuss their applications in our work in the following chapter.

Next, we will discuss previous papers relevant to sports player identification. From our literature review, there has only been a handful of works done on this topic. Out of the available literature, most of them focus on soccer and basketball players while there exists little or no work done for ice hockey. As far as we know, our system is the first to solve the problem of player identification in NHL hockey broadcast videos.

In our player identification problem, we have multiple consecutive bounding boxes of a player to consider. Similar to our approach, multiple frames are also considered in video classification tasks in order to model the temporal dynamics in a video. Consequently, we can draw resemblances between the nature of our problem and that of video classification. There are numerous successful works related to video classification, some of which inspired our own approach. Thus, we will present previous literature that tackle various video classification problems, including activity recognition and video-based person re-identification.

2.1 Background

2.1.1 Convolutional Neural Networks

Convolutional neural networks (CNNs) have achieved significant progress in recent years and have become the backbone in almost all approaches to visual image recognition problems due to their performance. Their popularity grew after CNNs were implemented by [7] to obtain state-of-the-art results on the ImageNet Large-Scale Visual Recognition Challenge 2012 [8] (ILSVRC-2012). Since then, further improvements to network architectures and computer hardware have led to better performance and made even deeper CNNs a possibility.

CNNs are a sequence of layers that transform an input image into an output vector of final class scores through a series of different layers and elementwise activation functions. The most fundamental layer in a CNN is the convolutional layer, which is responsible for the computationally intensive task of learning the spatial structure of images through the use of a set of K learnable filters. These filters are the convolutional layer's parameters and are small 3-dimensional matrices ($F \times F \times d$) with a spatial extent F and depth d equal to that of the input volume. Given an input volume ($w \times h \times d$) to the convolutional layer during a forward pass, each filter convolves across the width w and height h of the input volume at a predetermined stride S. Through the convolution process, each filter creates a unique 2-dimensional activation map that contains the filter responses to the input volume. By the end of the process, these activation maps are stacked together to form the convolutional layer's output volume with a depth of K.

Other common layers in a CNN include pooling and rectified linear unit (ReLU) layers, which are typically implemented after convolutional layers. These layers apply a fixed function on the input volume, whereas the convolutional layers implement operations that are functions of the input volume and layer parameters. Pooling layers decrease the number of parameters and computation in a network to avoid overfitting by using the max pooling operation over regions in each activation map in the input volume to reduce its spatial dimensions. ReLU layers perform an elementwise nonlinearity activation function defined as $A(x) = max(0, x)$ to maintain a threshold on the output at 0 to produce a sparse activation. Other activation functions employed in CNNs include the tanh and sigmoid function, but their usage has declined in favor of the less computationally expensive ReLU function. At the end of a CNN is a fully-connected layer that outputs the final 1-dimensional vector of class scores, where the size of the vector is equal to the number of class labels. As the name suggests, every activation in a fully-connected layer is connected to all the activations in the input volume by using filters of equal dimensions to the input volume.

2.1.2 CNN Architectures

In this research, we explored several different CNN architectures to compare their performance in extracting features of jersey numbers. The CNN architectures available in the literature include the standard CNN [3], Residual Networks [1], and Densely Connected Convolutional Networks [2].

Standard CNN. The CNN base model, implemented in the Long-term Recurrent Convolutional Network (LRCN) [3] architecture, acts as a feature extractor to generate fixed-length vector representations from visual inputs. Architecturally, the layout of this CNN is nearly identical to AlexNet [7] and the CaffeNet [9] reference model and also shares similar hyperparameters with the network used in [10]. The CNN contains 8 layers: 5 convolutional layers and 3 fully-connected layers. In the first convolutional layer, a 227x227 cropped input image is convolved with 96 different 7x7 filters with a stride of 2 pixels to produce 96 feature maps. These feature maps are passed through a non-saturating nonlinearity function called Rectified Linear Units (ReLUs). The outputs of the ReLUs are then summarized by a max pooling layer and normalized by a response-normalization layer to aid in generalization. Similar operations occur in the second convolutional layer. In convolutional layers 3,4, and 5, the convoluted feature maps only pass through a ReLU layer. In layer 5, the feature maps pass through a final max-pooling layer and then through the 3 fully-connected layers. The network output feds into a softmax function, which generates a confidence score for the predicted class.

Residual Network. Residual Networks [1] (ResNets) address the complication of training deep neural networks. As the number of layers (depth) increases, a network becomes more difficult to optimize during training due to a problem of training accuracy degradation. As verified in [1], an increase in layers in a network leads to a higher training error, not caused by overfitting. The authors address this issue with a deep residual learning framework. In this framework, a desired underlying mapping $H(x)$ is optimized to be fit by a given stack of layers, where x

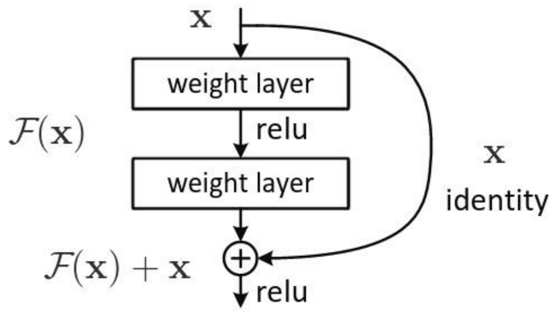


Figure 2–1: A layer stack with residual learning [1].

denotes the input to the stack. The authors hypothesized that if multiple nonlinear layers can asymptotically approximate intricate functions like $H(x)$, then the layers can asymptotically approximate the residual function $F(x) = H(x) - x$ with greater ease. The original mapping is then reformulated as $H(x) = F(x) + x$ where the residual function can be explicitly fitted. "Shortcut connections" are utilized to skip a stack of layers and compute the identity mapping of input x, as shown in Figure 2–1. Element-wise addition is then performed on the identity mapping and output of the skipped stacked layers.

One of the ResNet models we experimented with has 50 layers (ResNet-50). Each stack of layers contains 3 layers of 1x1, 3x3, and 1x1 convolutions respectively, which creates a "bottleneck" design for the 3x3 layer. The first 1x1 layer decreases the dimensions of the input into the 3x3 layer and the second 1x1 layer restores the original dimensions by increasing the dimensions of the output from the 3x3 layer. This "bottleneck" design minimizes the dimensions of the input and output of the 3x3 layer to reduce computation time during training. The other model we experimented with is a 10-layer ResNet [11] (ResNet-10), which differs in that it consists of four

2-layer stacks, rather than the 3-layer stacks. The 2-layer stacks do not utilize a "bottleneck" design since the 2-layer stacks are implemented in a much shallower network where training time is not a large concern. The ResNet-10 also replaces the mean subtraction during training with an additional batch normalization layer on the input data.

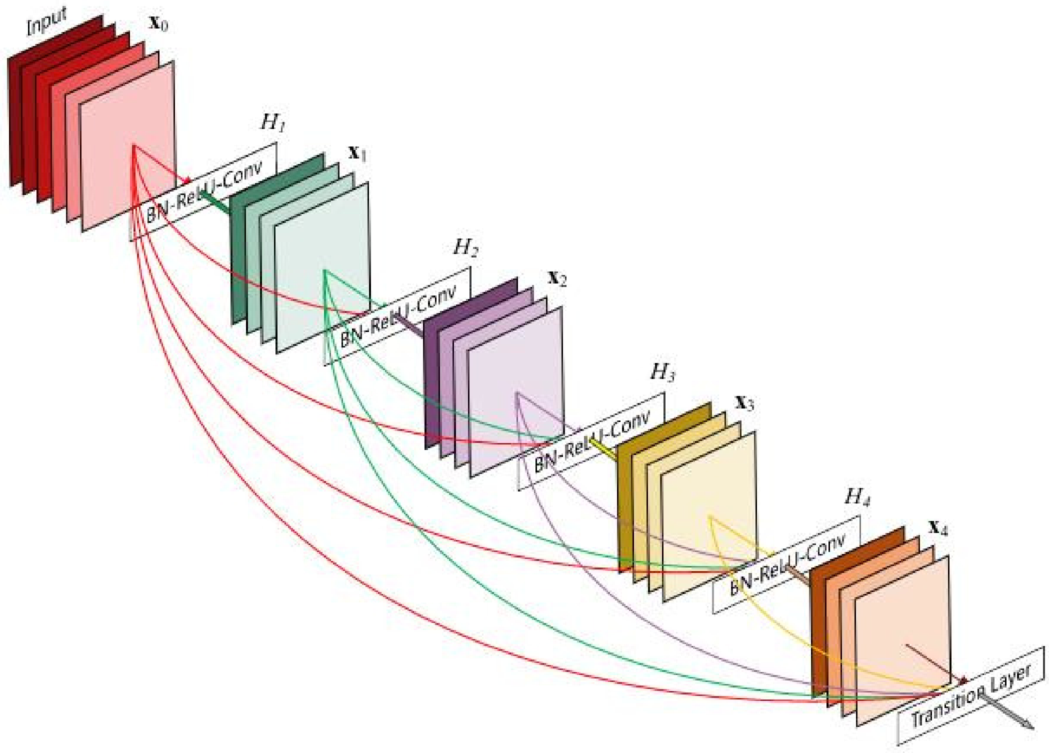


Figure 2–2: Connectivity pattern of a 5-layer DenseNet block [2].

Densely Connected Convolutional Networks. Densely Connected Convolutional Networks [2] (DenseNet) have a densely connected architecture that contains direct connections between all layers to maximize the information flow between each other. These direct connections are computable by maintaining the same feature map sizes for all layers in the network. In this network, each layer receives feature map outputs from all preceding layers and feeds its feature map outputs into all subsequent layers. With this connectivity pattern, the Ith layer has inputs from I layers and outputs to $L - I$ layers. As shown in Figure 2–2, the input for the lth layer is given as $x_I = H_I([x_0, x_1, ..., x_{I-1}])$, where x_0, x_1,..., x_{I-1} are the feature maps from preceding layers. In a L-layer DenseNet, there are $L(L + 1)/2$ connections, as opposed to traditional feed-forward architectures, which have L connections.

The DenseNet model [1] we used contains 121 layers and was converted from the original Torch model [2] into a Caffe format.

2.1.3 Recurrent Neural Networks

Recurrent neural networks (RNN) were first introduced in the literature [12, 13] around the 80's. RNNs were developed as a method for modeling sequential data where data points occur over a time series. In one of the first works [12] on RNNs, the authors demonstrated the application of the backpropagation learning rule in recurrent networks to model a sequence input. They theorized that a recurrent network can achieve the same behavior on a sequence input as a feedforward network that was duplicated at each time step of the sequence. The advantage of the recurrent network is that it does not require the extra hardware to store duplicates of itself as the feedforward network does.

[1] https://github.com/shicai/DenseNet-Caffe

Due to its recurrent nature, RNNs are used to model the temporal dynamics of a sequence input by mapping the sequence input to the hidden state and then mapping the hidden state to the output. The hidden state serves as the memory of the RNN and retains information from previous inputs. Connections between hidden states over different timesteps allow the network to accumulate past information over a finite period of time. As information accumulates, the hidden state becomes updated at each time step. This enables the network to discover temporal patterns between time steps that can influence the output at each time step. Compared to RNNs, traditional neural networks differ in that they assume the inputs to be independent of each other and produce outputs that are independent of previous inputs.

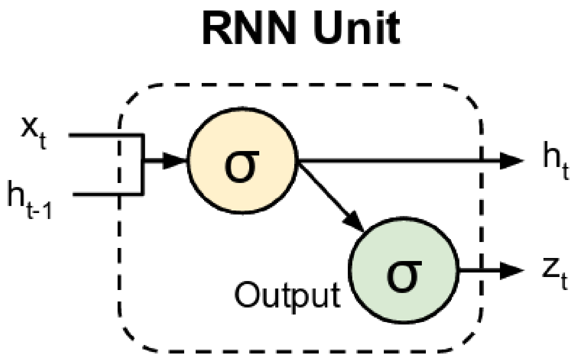


Figure 2–3: A diagram of an ordinary RNN unit [3].

If we were to unroll a RNN, we would end up with a series of RNN units, as shown in Figure 2–3, where each unit corresponds to a time step. At a given timestep t, the RNN unit takes in an input x_t and the previous hidden state h_{t-1} and maps them to the current hidden state h_t. The current hidden state h_t is then mapped to the output z_t of that time step. The following equations [3] summarizes the RNN mechanics:

$$h_t = g(W_{xh}x_t + W_{hh}h_{t-1} + b_h) \tag{2.1}$$

$$z_t = g(W_{hz}h_t + b_z) \tag{2.2}$$

The parameters of the RNN are given by the input weight W_{xh}, the recurrent weight W_{hh}, the output weight W_{hz}, and bias b_h and b_z. An element-wise non-linearity function g, usually a sigmoid or hyperbolic tangent, is used to perform the mapping.

In theory, the ordinary RNN is an ideal model for processing sequential data, but they are difficult to train in practice for learning long-term dynamics. This problem is attributed to occurrences known as the exploding gradient and vanishing gradient, which were first introduced in [14]. The exploding gradient occurs when the norm of the gradient increases exponentially during backpropagation training. On the other hand, the vanishing gradient occurs when the norm of the gradient decreases exponentially during backpropagation training. These occurrences are inherent in RNNs due to their recurrent nature. The gradient problems transpire when the gradient is propagated through many network layers over a long-range temporal interval. Later on, [15] showed that the exploding gradient problem can be mitigated by clipping the gradient at a maximum threshold. However, to alleviate the vanishing gradient issue, [6] proposed the long short-term memory (LSTM) to ease the difficulties of training RNNs mentioned in [14].

2.1.4 Long Short-Term Memory

Long short-term memory (LSTM) was first introduced by [6] as a solution to the problem of vanishing gradients in RNNs. Compared to an ordinary RNN, the LSTM differs in that its structure consists of gating mechanisms that allow each LSTM unit (or cell) to regulate the flow of information into and from itself. These mechanisms include the input gate i_t, forget gate f_t, output gate o_t, and input modulation gate g_t, as shown in Figure 2–4. Incorporating the gates into the structure of the LSTM cell enables the LSTM network to learn when to "forget" previous information and when to update itself with new information. Similar to ordinary RNN units, a LSTM cell has a hidden state h_t, which is often referred to as the output of the LSTM cell. In addition to the hidden state h_t, the LSTM cell also maintains a cell state c_t, which acts as the memory storage. The cell state c_t enables the LSTM to accumulate information over a long period of time to learn long-term temporal dynamics and is influenced by the previous hidden state h_{t-1} and input x_t.

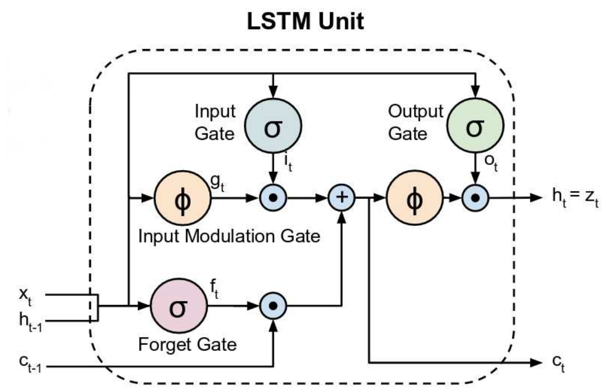


Figure 2–4: A diagram of a LSTM unit/cell used in this research and also used in [3] and [4].

For a more detailed understanding on the mechanics of a LSTM cell, we examine the following forward pass equations [3] below that summarize the mechanisms of the gates and behavior of the hidden state and cell state for a given timestep t. This LSTM variant was first described in [4] and is a simplified interpretation derived from the original LSTM architecture [6].

$$f_t = \sigma(W_{xf}x_t + W_{hf}h_{t-1} + b_f) \tag{2.3}$$

$$i_t = \sigma(W_{xi}x_t + W_{hi}h_{t-1} + b_i) \tag{2.4}$$

$$o_t = \sigma(W_{xo}x_t + W_{ho}h_{t-1} + b_o) \tag{2.5}$$

$$g_t = tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_c) \tag{2.6}$$

$$c_t = f_t \otimes c_{t-1} + i_t \otimes g_t \tag{2.7}$$

$$h_t = o_t \otimes tanh(c_t) \tag{2.8}$$

At each timestep, the LSTM cell performs several element-wise computations via the gates to update its cell state c_t and hidden state h_t. Based on the equations above, the gates are all dependent on the input and previous hidden state. Since the forget gate f_t, input gate i_t, and output gate o_t are sigmoidal functions, their outputs are values within the range [0,1]. The values of these three gates determine the amount of information that goes into the cell state ct and hidden state h_t. The last gate, the input modulation gate g_t, is a hyperbolic tangent function that activates the contents of the input x_t and previous hidden state h_{t-1} and provides a vector of candidate input values. To update the cell state c_t and store information into the memory, the forget gate f_t first determines what information should be selectively "forgotten" from the previous cell state c_{t-1}. Next, the input gate it determines what information from the input modulation gate g_t should be selected and transferred into the cell state c_t. Thus, the cell state ct can be expressed as the sum of the previous cell state c_{t-1} and the input modulation gate g_t, which are regulated by the forget gate f_t and input gate i_t respectively. Once the cell state c_t is updated, the output of the LSTM cell, or hidden state h_t, can be determined. The hidden state h_t is a hyperbolic tangent function of the current cell state c_t, which is regulated by the output gate o_t to decide what information should be transferred from the cell state c_t to the hidden state h_t.

The additive nature of updating the cell state c_t, as presented in the equation, is a key element in the design of the LSTM that alleviates the vanishing gradient concern. During backpropagation, when the derivative of the cell state c_t is taken with respect to the previous cell state c_{t-1}, the added input term $i_t \otimes g_t$ is a constant and becomes zero. The sum operation with the input term preserves a constant error to flow back through time and still allows the cell state c_t to be updated with new inputs through addition. The absence of multiplication prevents the gradient of the cell state c_t from approaching zero or vanishing if $i_t \otimes g_t < 1$.

Yet, the forget gate f_t still relies on multiplication to modulate the previous cell state c_{t-1}. Since the forget gate f_t is a sigmoidal function, its derivative is less than 0.25 and multiplying by a value < 1 recurrently will result in the vanishing gradient, as is the case in ordinary RNNs. However, the purpose of the forget gate f_t is to learn what information to "forget" or to store. When the forget gate f_t is on or has an output of 1, it transfers all the information from the previous cell state c_{t-1} to the current cell state c_t by computing the identity function. The identity function always has a derivative of one and a forget gate f_t that is on will allow the cell state gradient to backpropagate through time unchanged and resist vanishing.

2.2 Sports Player Identification

There are a few works in the literature that attempt to solve the problem of automatic sports player identification in broadcast sports videos. The majority of these works developed their approaches for videos of soccer and basketball games. To the best of our knowledge, our proposed method is the first to be implemented on NHL hockey broadcast videos.

Most of the earlier computer vision approaches rely on close-up shots of players in a broadcast video feed. This is a reasonable approach since close-up shots offer the most clarity on a player's features, including their faces and the text on their jerseys. The majority of these earlier works follow similar methods of using optical character recognition (OCR) software to distinguish players by their jersey number. Several methods involve facial recognition [16, 17] or image segmentation [18, 19, 20] to first detect and isolate the player from the rest of the frame before extracting text from the jersey for the OCR.

In [16], Bertini *et al.* propose a player identification method for soccer videos by first detecting the frontal faces of soccer players in close-up shots and then matching them with their identity. The player's identity can be derived from the textual cues of numbers on the jerseys or superimposed text captions in the video frame. The face and jersey number detection task is accomplished by a detection algorithm [21] that depends on multiple classifiers to detect the presence of simple features. Numerous detectors are trained to classify each specific jersey number and player's face. The training data includes both positive and negative samples that were manually cropped from video frames. For players that are detected but not identified due to a lack of textual cues, the detected face is matched with previously identified faces by using a face similarity measure. Later in [17], the same authors improved upon [16] by introducing a new method of reading textual cues, which involves a combination of interest region detectors. A Harris corner detector is first used to extract image corners and an unsupervised clustering process is applied to the corners to determine regions of interest. Maximally stable extremal regions [22] (MESER) are then extracted from the regions of interest to further improve the selection of candidate text regions for processing with a commercially available OCR.

In [20], Ye *et al.* experiment on close-up shots of a variety of sports and use image segmentation to first separate the jersey numbers of the players from the background. A generalized learning vector quantization (GLVQ) [23] is used to segment candidate regions that may contain the jersey number by clustering image pixels into limited colour-homogenous regions. These regions are then filtered to discard non-jersey number regions by considering size and shape attributes, including pipe-like components (PLC) [20]. To prepare the remaining candidate regions for classification, rotation, scale, and non-rigid deformations need to be considered. So Ye *et al.* use Zernike moments [24] as shape descriptors to represent their features since Zernike moments are invariant to rotation and scale due to the orthogonality. A k-nearest neighbour (k-NN) classifier is then trained on synthetic, non-rigid samples of number digits to alleviate the non-rigid deformation issue when classifying the candidate segmented regions into one of the number classes.

The player identification method in [19] also performs image segmentation in close-up shots by considering the color contrast between the jersey and number. Since the color contrast is usually significant in one component of the HSV color space, image segmentation can be done by thresholding that component to separate jersey regions. The result of segmentation is a bitmap where the numbers are distinguished from the jersey as “holes”. The spatial relation between the “hole” regions and the rest of the bitmap allows candidate number regions to be extracted by detecting the internal contours. False candidates are filtered according to the area and aspect ratio of the candidate regions. Remaining candidates are smoothed by median filtering and their character rotations are normalized based on the central contour moments and the candidates are then processed by a standard OCR. To improve the OCR results, temporal redundancy is taken into account to ensure the number predictions are consistent across consecutive frames.

Messelodi and Modena propose a method in [18] to identify athletes in raw footage of track-and-field events where the camera view ranges from close-up shots to long shots. A text extractor algorithm [25] is applied to extract text from athletes’ bibs by focusing on unsaturated color zones, which best correspond to the black and white areas of the bibs. The extracted text is then fed into an OCR to produce string representations of the text. This method also makes use of prior knowledge of possible athletes by comparing the identity prediction with a list of expected candidate identities to produce a confidence score. Although this approach works well for close-up shots, it is less successful in long shots where the text on the bibs are frequently missed by the text extractor.

Some of the more recent works like [26] attempt to identify players from the field-view shots of sports broadcast videos, rather than relying on close-up shots. These field-view shots are recorded by a single moving camera that pans across the playing field to provide viewers with an encompassing view of the game and the players. Compared to the close-up shots that the previously mentioned works performed on, the field-view of broadcast videos provide far less details of the players' features due to their constant motion and pose variation. Adopting the earlier approaches of using an OCR to read text in the field-view of broadcast videos would perform poorly given the difference in clarity of the features [27]. Instead of employing an OCR, recent works model a feature combination of MSER [22], SIFT [28], and color features of players [27, 29] and train deep generative architectures like convolutional neural networks [30, 31].

The approach in [27] is one of the first published works to perform both player tracking and identification in broadcast videos. The authors, Lu *et al.*, experiment with NBA basketball footage to implement joint probabilistic inference about player identities with a conditional random field [32] (CRF) model. They first generate player appearance models by combining three different features, including MSER [22], SIFT [28], and RGB color histograms, to create visual words. Detections of the same player across multiple frames are connected with temporal edges. The combined features from multiple images are then used by the CRF to infer the player's identity using loopy belief propagation [33]. The same authors later obtained better accuracy results [29] by proposing an additional Linear Programming Relaxation algorithm to simplify the inference optimization problem.

Similar to [27], another method [26] propose a player identification framework for basketball players in NBA broadcast videos. This paper utilizes a deformable part model [34] (DPM) to detect players by using template matching to approximate the shapes of the players. Once detected, gradient difference is used to extract the jersey number from the player. However, the authors still relied on an OCR to recognize the number.

In one of the first papers [30] to propose a deep learning approach, Gerke *et al.* use a deep convolutional neural network (CNN) to perform pixel-to-jersey number learning. The CNN is trained on 10,000 images cropped from soccer broadcast videos of players with clearly visible jersey numbers. The authors compare two approaches of classifying the jersey numbers. In the first approach, each number is modelled as a separate class. Alternatively, in the second approach, each of the two digits in a number is a separate class. Their results show that the more holistic approach of considering each number as a separate class performs better. In a more recent work [31] by the same authors, Gerke *et al.* improve upon their player identification system by fusing the original deep learning approach [30] with a spatial constellation feature that is based on the positions of the players of a team. The two modalities are combined using cost matrix fusion. Each modality yields a cost matrix that is row-normalized to represent a probability distribution and the final weighted cost matrix is calculated to deliver a single optimal assignment.

Most of the early sports player identification methods rely heavily on close-up shots of players and OCR software to recognize any text to determine the players' identities. However, these approaches are not suitable if we wish to label players for the entire duration of a game, since the camera defaults to the field-view for the majority of the game. To identify players from a field-view camera, other approaches [27, 29] model players based on their appearance features, but they rely too heavily on color and shape. Also, without segmenting the players from the background, some features are generated from the background, which makes noisy features. A deep learning approach [30] seems promising, but the authors only perform player identification on single frames. To perform player identification in videos, we require a method that can observe players over time and operate on sequential data. Thus we will look into previous work on video classification in the next section.

2.3 Deep Neural Networks for Video Classification

In this section, we present previous literature related to the computer vision problem of video classification. In a typical video classification problem, the goal is to acquire global video descriptors that encode both the spatial information in the video frames and the dynamic temporal information from the motion that occurs across consecutive frames. Similarly in our player identification problem, we aim to encode sequences of bounding box images of a player by considering the spatial and temporal features of the jersey number as its appearance changes due to the player's movements.

The traditional video classification approach [35, 36, 37, 38] is based on the encoding and classification of spatio-temporal features. These algorithms first detect local visual features and extract them as a set of sparse or dense interest points to describe specific regions in a video. The interest points can be selected with a feature detector and then characterized by spatio-temporal feature descriptors (i.e. HOF/HOF [39] and Cuboid [40]). The features can then be encoded into a bag-of-words representation to generate a fixed-sized global video-level descriptor. Finally, a classifier, like a SVM, can be trained on the bag-of-word representations to learn the distinctions between different visual classes.

While large-scale image datasets and advanced GPU hardware helped boost the development of deep learning in the image classification domain, it was not until recently that the application of deep learning was extended to the video classification problem. Encouraged by the positive results of CNNs and the recent introduction of more large-scale annotated video datasets, such as the UCF101 action recognition dataset [41] and the massive YouTube-8M dataset [42], video classification has started to gain popularity as a research area. With larger video datasets available, training deep architectures on videos finally became feasible. Since the majority of these video datasets feature human activity, action recognition became one of the more common research areas of video classification in the literature.

Generally, CNNs are limited to handling only 2D inputs, with images being the dominant example. One method of applying CNNs to video classification is to extend an image CNN to the video domain and treat the space and time dimensions as equal inputs. To do this, several works in the literature [43, 44, 45, 46, 47] propose 3D convolutional neural networks for performing 3D convolutions to multiple frames to model the spatio-temporal features in videos. In [47], Karpathy *et al.* concludes that while CNNs are already capable of demonstrating high performances on individual video frames, a 3D CNN that models additional motion cues can also perform just as well, but with little improvement. This may indicate that properly modeling motion cues is a difficult task or that the learnt spatial-temporal features do not represent the motions well [48]. Taking this issue into account, Simonyan *et al.* [48] propose a two-stream CNN model that decomposes a video into spatial and temporal components and dedicates a computational stream to each one. The temporal stream learns the motion information by operating directly on the optical flow from multiple frames, which proves to perform significantly better than CNNs that train only on raw video frames. However, these approaches only sample short video clips and can be limited by the small time periods.

Other works on video classification try to learn spatio-temporal features over longer sequences by employing recurrent networks. In particular, the LSTM variant of recurrent networks has been proposed for video classification tasks by several works due to its prior success in other tasks and ability to mitigate the problem of vanishing and exploding gradients. The approach in [49] was one of the first to apply LSTMs to video classification by training the model on a bag-of-words appearance descriptor and SIFT features that describe the motion. Some of the more recent works [50, 51, 52, 3] incorporate both CNNs and LSTMs to learn spatial and temporal cues respectively. Donahue *et al.* [3] propose the long-term recurrent convolutional network (LRCN), an end-to-end trainable network that combines a standard CNN with a layer of LSTMs. The LRCN showed improvements over single-frame models on action recognition tasks by considering both raw image and optical flow inputs. In [50], Ng *et al.* introduce a similar model that computes CNN features on optical flow and raw frame images and uses a five-layer LSTM to learn a global description of the video from sequences of CNN activations. Ng *et al.* demonstrate that compared to feature pooling networks, a more sophisticated temporal model like LSTMs can benefit more from optical flow to produce better results. The hybrid framework [51] proposed by Wu *et al.* also extracts long-term spatio-temporal information by incorporating the two-stream CNN approach [48] and feeding the CNN activations into a LSTM network. Additionally, Wu *et al.* adopts a regularized feature pooling network to fuse short-term video-level spatial and temporal features together, which is then combined with the outputs of the LSTM to generate a final prediction. By combining both frame-level and video-level features, Wu *et al.* achieve state-of-the-art results on the UCF101 dataset [41]. In [52], the same authors employ a third CNN stream to exploit audio cues in videos, which results in a slight increase in performance compared to [51].

Similar to the state-of-the-art video classification methods, several recent approaches [53, 54, 55] also integrate recurrent units into their model to learn spatio-temporal cues for video-based person re-identification tasks. To perform person re-identification in the video setting, a video recording of a person in one camera needs to be matched, often by the person's shape and/or color, with a gallery of videos of the same person recorded by different cameras from different angles. This problem is comparable to our player identification problem, but in our situation, the player's shape and/or color is not enough to differentiate an individual player since players of the same team wear the same jersey. McLaughlin *et al.* [53] is the first to introduce a recurrent convolutional network architecture similar to [50, 3] for person re-identification. Their architecture uses a CNN to represent the appearance of the person in each frame and the recurrent layer allows information to flow between time-steps. The outputs from each time-step are combined by temporal pooling and then fed into a Siamese network [56] to perform re-identification. Wu *et al.* [54] also propose a model similar to [53]. Instead of using traditional LSTMs for the recurrent layer, the authors adopt the Grated Recurrent Unit [57] (GRU) architecture, which is a variation of the LSTM formulation and requires fewer parameters. Re-identification is performed by using a similarity function defined by the inner product of the outputs of the recurrent layer of a video pair. In [55], Yan *et al.* propose a recurrent feature aggregation network (RFA-Net) that extracts hand-crafted features, including local binary patterns [58] (LBP), HSV and lab color channels, from video frames. These features are then aggregated using a LSTM layer, as implemented in [3], and concatenated to produce a video-level representation. The RankSVM method [59] is then used to compute the similarity of the video pair.

2.4 Summary

Our literature review shows numerous works that apply deep learning approaches with recurrent operations to different kinds of sequential data, including videos of human activity and people. Most of the recent papers show promising results and state-of-the-art performance, particularly when the models exploit the temporal nature of videos. Encouraged by these positive results, we aim to deliver a similar end-to-end trainable approach to operate on hockey player tracklets and perform hockey player identification. Our architecture combines CNN and LSTM layers to learn features of the players' jersey numbers of any variable length player tracklet and perform temporal pooling to produce an overall confidence score for the tracklet.

CHAPTER 3
Methodology

In this chapter, we present the approach we take to develop a deep neural network to accurately identify hockey players in NHL broadcast hockey videos. We discuss in detail about the datasets that are used in the training process, the different network architectures that are experimented with, and the training methodology.

3.1 Approach

In this work, we develop a method for identifying hockey players in National Hockey League (NHL) broadcast hockey videos using a deep learning approach. Our approach is a two-step process. In the first step, we design an end-to-end trainable deep neural network model that combines a convolutional neural network (CNN) with a long short-term memory (LSTM) layer. We propose this model, which we refer to as a CNN+LSTM network, to process a sequence of input images of a player and output a confidence score for each input image. In the second step of our approach, we combine the confidence scores from the entire image sequence with a so-called late fusion technique to classify the image sequence with one label. The late fusion technique involves averaging all the confidence scores of an image sequence and inputting the average score into a secondary CNN classifier to generate the label prediction. In this work, we label players by their jersey number. The images, with which we train and evaluate our CNN+LSTM network, are bounding boxes of hockey players cropped directly from NHL broadcast hockey videos. With this network, we identify players solely based on their visual features without taking into consideration any priors, such as game or player context. For example, a player's time-on-ice information is a context that would allow us to know exactly when the player was on the ice at any given moment during a game. This context can help narrow down the list of possible candidates when performing player identification, but this data is not always available.

The use of hockey videos allows us to determine the identity of hockey players by making multiple observations over a period of time. Since a player's image is captured by a video camera, we have an entire set of video frames to observe the player at multiple instances. For the purpose of this research, we only consider sections of the NHL broadcast videos that are recorded by the main camera in the stadium. The main camera is most commonly used during NHL hockey games since it pans across the hockey rink to provide viewers an encompassing view of the game and the players.

From the view of the main camera in NHL broadcast videos, hockey players have very similar appearances, particularly when they belong to the same team. Distinguishing individual players by facial or physical body features is often extremely difficult to accomplish, due to the limited frame resolution of the camera. The most prominent visual feature we can observe is the jersey number on the back of every player's jersey. However, the jersey number is not always visible to the camera's view due to occlusion, motion blur, and the angle in which the players are facing the camera. Considering that the jersey number is not frequently visible, it will be difficult to identify the player if we are only given a single frame individually to make our observations. However, if we observe a player over a period of time, we should be able to combine the observations across multiple frames and correctly identify the player's jersey number. There are also numbers on the sleeves of the jerseys in addition to the number on the back, but these numbers are often too small to be visible from the main camera's view.

In order for us to observe a player over a period of time to extract the visual features of the player's jersey number from multiple video frames, we must first be able to locate the player in all the frames. We acquired SPORTLOGiQ's ground truth player tracking data to obtain player detections in every frame of recorded NHL hockey broadcast videos. The ground truth data was used in this thesis to build and evaluate different models. We used the available player tracking data to locate the players in every video frame and mark their location with bounding boxes, as shown in Figure 3–1. The estimated detections were associated with similar detections across other frames using the Hungarian algorithm [60] to infer short sets of trajectories (or tracklets) for every player visible in a frame. We ended up with tracklets that can each be considered as a sequence of bounding boxes that contained the same player over a duration of time. Each player tracklet's starting point occurred when a player was first detected in the main camera's field of view and the tracklet ended when the player was no longer present in the video. We cropped these bounding boxes directly from the video frames to create sequences of images for each player. We annotated and categorized these sequences into a new dataset, the NHL Hockey Player Tracklet (NHL-HPT) dataset, to train and evaluate our model. More details on the tracklet creation process will be in the *Chapter 3.2 Datasets* subsection and additional details on the player tracking data will be in the *Chapter 3.2.2.1 Dataset Generation* subsection.

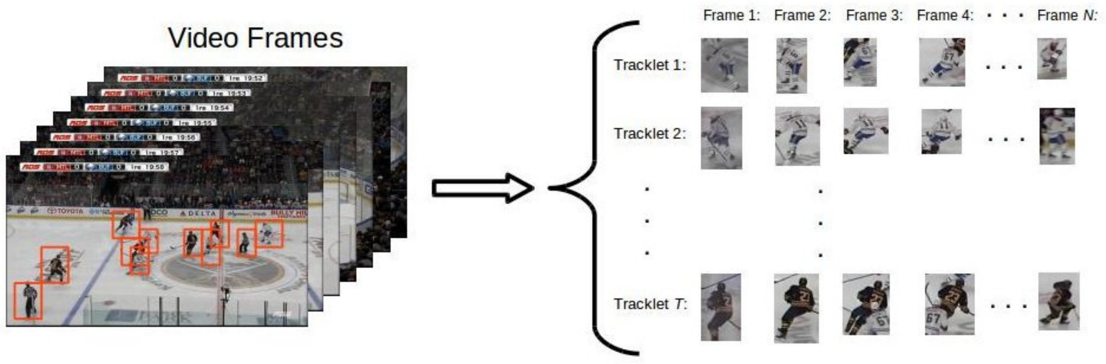


Figure 3–1: Player tracklets are sequences of bounding boxes that contain the same player over a duration of time. The bounding boxes were cropped directly from NHL broadcast hockey video frames. The video frames pictured are from the main camera of the stadium.

In our first set of experiments, we began by exploring different CNN architectures to determine the most suitable architecture to form the CNN base of our CNN+LSTM network. We evaluated different CNNs on the SPORTLOGiQ NHL Hockey Player (NHL-HP) dataset, which contains single still images of hockey players with the jersey number fully visible. From these experiments, we chose a CNN that can successfully extract useful visual features from the jersey number within a reasonable training and execution time. Since very deep networks can take over a week to train, we wanted a network that can be trained within a couple days at most to ensure we do not surpass the project timeline and still achieve comparable performance.

Once we have determined the specific CNN architecture to implement in our CNN+LSTM network, our next set of experiments involved training and evaluating the CNN+LSTM network on the new NHL Hockey Player Tracklet (NHL-HPT) dataset. As mentioned earlier, this dataset contains sequences of bounding box images of players, which the recurrent part of the network can use to learn any temporal dynamics in the sequence. As opposed to a single still image, a tracklet increases the chances of detecting the correct jersey number of the player since it provides multiple images of the player. Given a tracklet as an input, the CNN+LSTM network outputs a confidence score for the label prediction of each bounding box image in the tracklet. To predict the label for the overall tracklet, we used a so-called late fusion approach to combine the confidence scores of every bounding box. One late fusion method is to average the confidence scores across every bounding box and determine the label with the maximum probability. We wished to further improve the accuracy of our approach, so we took the late fusion method a step further and treated the averaged confidence scores of each tracklet as a feature vector input for a secondary CNN classifier. The secondary CNN classifier can recognize certain patterns within the confidence scores themselves to generate a more accurate label prediction.

3.2 Training and Test Datasets

There are currently no *publicly* available datasets with labeled images of sports players. Since our goal is to identify players directly in broadcast hockey videos, we require image data that originate from the broadcast videos to properly train and evaluate our model. The data should be images of hockey players cropped from the broadcast video frames and should be annotated by the player's jersey number. We were able to obtain access to the SPORTLOGiQ NHL Hockey Player (NHL-HP) dataset for testing CNN architectures. We were also able to obtain access to the asset of recorded NHL broadcast hockey videos and their respective player tracking data and annotations. We used the acquired resources and data to create a new NHL Hockey Player Tracklet (NHL-HPT) dataset. The usage of the NHL-HPT dataset is currently restricted to use only by SPORTLOGiQ and it allows us to evaluate our proposed method of player identification when given a player's tracklet, where occluded jersey number instances are common.

3.2.1 SPORTLOGiQ NHL Hockey Player Dataset

The SPORTLOGiQ NHL Hockey Player (NHL-HP) dataset is a collection of individual hockey player bounding box images annotated with the player's jersey numbers that range from 1 to 98 (number 99 has been retired league-wide to honor Wayne Gretzky). These bounding box images were cropped directly from broadcast hockey video frames. This dataset only contains player images where the jersey number on the back of the player is fully visible and identifiable, with no parts of the jersey number occluded or hidden. Occluded number instances were not included in this dataset since the purpose of this particular dataset was to train a CNN network to learn visible numbers. The occlusion problem will later be handled by our recurrent model. The dataset is divided into two sets, with 45,469 images for a training set and 3,885 images for a test set. This total also includes augmented data via cropping, padding and scaling. The dataset was used to train and test different

CNN architecture models to evaluate their performance in feature extraction for jersey number identification in broadcast videos.

3.2.2 NHL Hockey Player Tracklet Dataset

Our player identification approach makes use of videos to take advantage of the availability of sequences of images that is inherent in videos. To train and evaluate our approach, we require a dataset that contains sequences of images of players to simulate how we observe a player in a video. Since our focus is on identifying hockey players, we want to extract the players from the whole video frame on a frame-by-frame basis to produce sequences of smaller player-centric images. This can be done by enclosing the players in every frame with bounding boxes and cropping the boxes as images. Unfortunately, at the time of this work, there were no publicly available datasets that met the requirements we described in order to properly train and test our method. Hence, we created a new NHL Hockey Player Tracklet (NHL-HPT) dataset, which is a collection of player tracklets represented by sequences of player bounding box images. A **player tracklet** is defined as a fragment of a player's overall trajectory and consists of a bounding box detection of the player at every time-step of the tracklet. All the bounding box images associated in a tracklet are extracted directly from the broadcast video frames. Within a tracklet, each image is labeled with the same jersey number as the player that the tracklet corresponds to. The player's jersey number may not be visible in every image in a tracklet, but it should be visible in at least more than one image. The length of the tracklet can vary considerably from 16 to several hundred frames. Figure 3–2 shows examples of player tracklets that have visible jersey numbers.

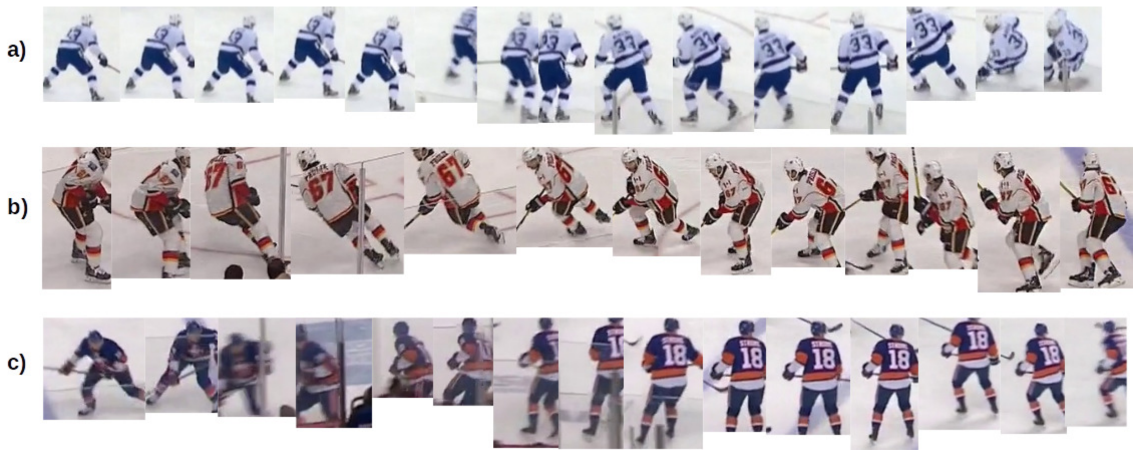


Figure 3–2: Shown are examples of player tracklets with visible jersey numbers. Tracklets are sampled from videos at 30fps. Above, the frames displayed are sampled from the tracklets at 3fps.

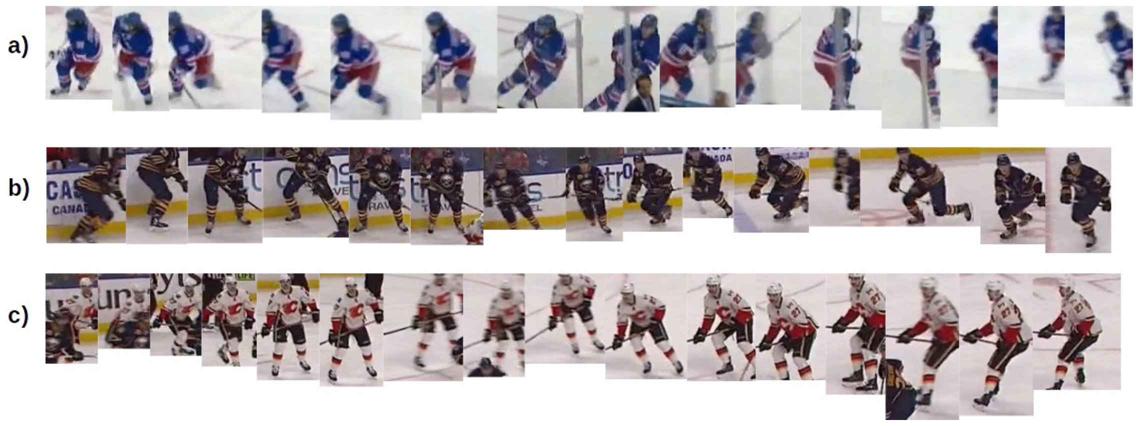


Figure 3–3: Shown are examples of player tracklets with non-visible jersey numbers. Tracklets are sampled from videos at 30fps. Above, the frames displayed are sampled from the tracklets at 3fps.

The dataset currently contains tracklet examples for 81 classes, which includes classes for 79 different jersey numbers, referees, and unidentifiable numbers. We reserve a label for tracklets that are unidentifiable if the jersey number is not visible throughout the entire tracklet, as shown in Figure 3–3. Oftentimes, a player's jersey number will not always be visible to viewers in the broadcast video, so assigning a label to confirm a player is unknown is more appropriate than classifying the player with an incorrect label.

The dataset currently does not cover all possible jersey numbers in the NHL as there was not enough data available during the creation of the dataset. It was possible to increase the number of examples with non-NHL jersey images, but there were no publicly available datasets that were suitable. We sorted the data by jersey numbers, so that each jersey number class contained images of players with different jersey uniform styles and colors. This was done to train our networks to recognize jersey numbers despite the different visual appearances of different hockey jerseys. We randomly divided the dataset into two sets [1] : a training set with 5,267 tracklet examples, and a test set with 1,278 tracklet examples. Due to the uneven availability of data for each jersey number, the number of examples for each class is not equally distributed. This resulted in an unbalanced dataset, as displayed in the class distribution plot of the training and test dataset in Figure 3–4. This created a challenge

[1] Since our dataset was limited, we only divided it into a training and test set without setting aside a validation set. As an alternative, we used k-fold cross validation in some experiments to detect over-fitting.

for our deep neural network to learn all the visual features that distinguish jersey numbers from each other.

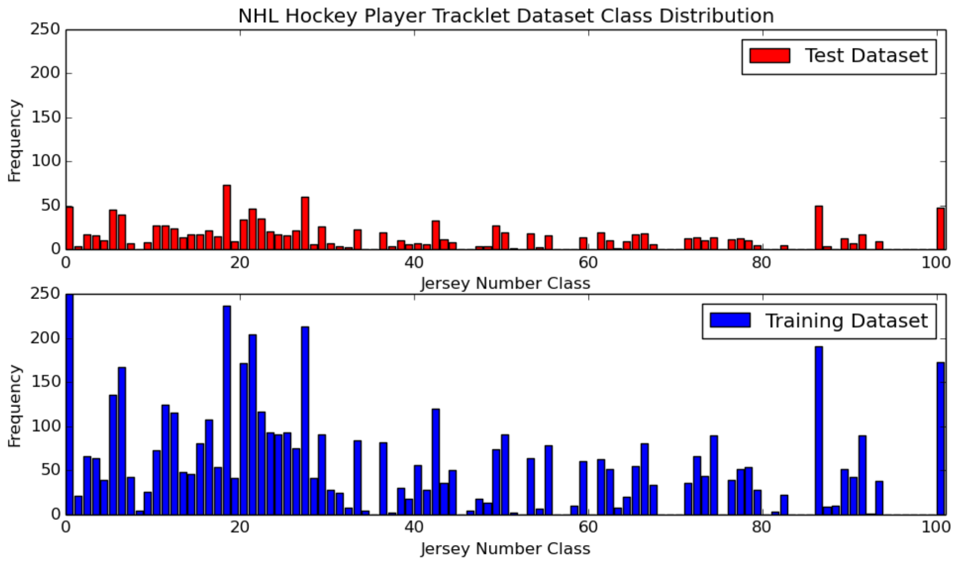


Figure 3–4: The plots show the distribution of number of examples (frequency) in each class in both the test and training subsets of the NHL Hockey Player Tracklet Dataset. The distribution is unbalanced due to the uneven availability of data for some jersey numbers. The label '0' is allocated for referees and the label '100' is for unknown player tracklets where the jersey number is not visible at all. All other labels correspond to the actual jersey number.

3.2.3 Dataset Generation

To create labeled tracklets for the new NHL-HPT dataset, we used the player tracking data generated by SPORTLOGiQ's player tracker to crop player bounding boxes from the broadcast hockey video files. The process of annotating the tracklets involved a series of manual and automatic labeling tasks.

The first step in creating our dataset was to perform tracking on all the players in a broadcast video using a tracking-by-detection framework based on the single shot multibox detector (SSD) [61]. The SSD is a convolutional neural network that outputs a set number of bounding boxes and the scores for the boxes to contain the existence of an object class instance. From the set of bounding boxes and scores, the final detections are decided by a non-maximal suppression step. The SSD was applied to every video frame to produce multiple hockey player detections as player bounding boxes, which describes the player's spatial pixel location in the frame.

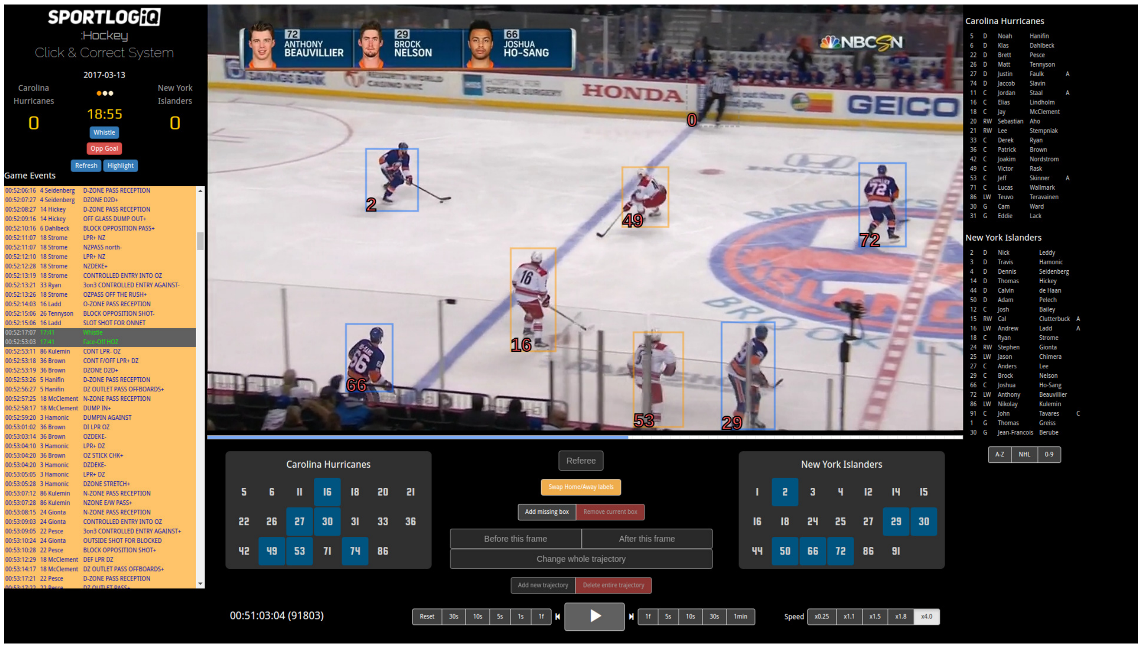
The SSD model uses a standard convolutional neural network architecture as the base network and adds additional convolutional feature layers to the end. These feature layers are used to generate a set of bounding box detections and their scores using a set of convolutional filters. In each video frame, the SSD overlays several feature maps with different grid scales (ie. 8x8, 4x4, etc.). At each location on the feature maps, a small set of bounding boxes of different aspect ratios is produced. The probabilities of a player being enclosed in each of the bounding boxes are generated and final bounding box detections are determined by non-maximum suppression. The final detections are then associated with detections from the previous frame to indicate that they belong to same player tracklet. If a current detection is not associated with a previous detection, the current detection becomes the start of a new tracklet. Tracklets end once the player is no longer visible in the next frame. This process was executed for every frame until the end of the video. Each video of a game period can generate approximately 5000 tracklets.

An issue with the player tracker was that errors may occur. When players overlap or occlude each other, a player bounding box detection can be assigned to the wrong tracklet. This was undesirable since each tracklet should only be associated with one individual player. Thus, to generate a ground truth dataset, correction was required to ensure that each tracklet only consists of bounding boxes of one player. This was done by using a "Click & Correct System" [2] web application that offers a graphical user interface to edit bounding boxes and player labels, as shown in Figure 3–5. In the "Click & Correct System", a broadcast video is displayed and a user is able to edit the bounding boxes around the players in each frame. The "Click & Correct System" was also used to manually annotate the player tracklets with their jersey number by clicking on the correct jersey number listed below the video. This correction was done for the entire duration of broadcast videos to create labeled ground truth player tracking data. After the correction, the tracking data for each broadcast video was saved into a JSON [3] data file format. A Python [4] script was written to parse the bounding box information from the JSON file and crop the bounding boxes directly from each frame in the video and store them as individual images.

[2] The "Click & Correct System" is a software developed by SPORTLOGiQ and only available internally.

[3] JSON is a language-independent data format that uses attribute-value pairs and array data types.

[4] Python is a popular open-source high-level programming language.



Game Events

```
00:52:06:16  4 Seidenberg   D-ZONE PASS RECEPTION
00:52:07:27  4 Seidenberg   DZONE D2D+
00:52:08:27  14 Hickey      D-ZONE PASS RECEPTION
00:52:09:16  14 Hickey      OFF GLASS DUMP OUT+
00:52:10:16  6 Dahlbeck     BLOCK OPPOSITION PASS+
00:52:11:07  18 Strome      LPR+ NZ
00:52:11:07  18 Strome      NZPASS north-
00:52:12:10  18 Strome      LPR+ NZ
00:52:12:28  18 Strome      NZDEKE+
00:52:13:19  18 Strome      CONTROLLED ENTRY INTO OZ
00:52:13:21  33 Ryan        3on3 CONTROLLED ENTRY AGAINST-
00:52:13:26  18 Strome      OZPASS OFF THE RUSH+
00:52:14:03  16 Ladd        O-ZONE PASS RECEPTION
00:52:15:06  26 Tennyson    BLOCK OPPOSITION SHOT-
00:52:15:06  16 Ladd        SLOT SHOT FOR ONNET
00:52:17:07  17:41          Whistle
00:52:53:03  17:41          Face-Off HOZ
00:52:53:11  86 Kulemin     CONT LPR- OZ
00:52:53:18  36 Brown       CONT F/OFF LPR+ DZ
00:52:53:19  36 Brown       DZONE D2D+
00:52:53:26  5 Hanifin      D-ZONE PASS RECEPTION
00:52:56:27  5 Hanifin      DZ OUTLET PASS OFFBOARDS+
00:52:57:25  18 McClement   N-ZONE PASS RECEPTION
00:52:58:17  18 McClement   DUMP IN+
00:52:59:20  3 Hamonic      DUMPIN AGAINST
00:53:01:02  36 Brown       DI LPR OZ
00:53:03:14  36 Brown       OZDEKE-
00:53:04:10  3 Hamonic      LPR+ DZ
00:53:04:20  36 Brown       OZ STICK CHK+
00:53:04:20  3 Hamonic      DZDEKE-
00:53:05:05  3 Hamonic      LPR+ DZ
00:53:05:28  3 Hamonic      DZONE STRETCH+
00:53:07:12  86 Kulemin     N-ZONE PASS RECEPTION
00:53:07:28  86 Kulemin     NZONE E/W PASS+
00:53:08:15  24 Gionta      N-ZONE PASS RECEPTION
00:53:09:03  24 Gionta      CONTROLLED ENTRY INTO OZ
00:53:09:05  22 Pesce       3on3 CONTROLLED ENTRY AGAINST+
00:53:10:24  24 Gionta      OUTSIDE SHOT FOR BLOCKED
00:53:10:28  22 Pesce       BLOCK OPPOSITION SHOT+
00:53:12:29  18 McClement   DEF LPR DZ
00:53:14:17  18 McClement   DZ OUTLET PASS OFFBOARDS+
00:53:17:21  22 Pesce       D-ZONE PASS RECEPTION
00:53:17:22  22 Pesce       DZ OUTLET PASS+
```

Carolina Hurricanes

5	D	Noah	Hanifin	
6	D	Klas	Dahlbeck	
22	D	Brett	Pesce	
26	D	Matt	Tennyson	
27	D	Justin	Faulk	A
74	D	Jaccob	Slavin	
11	C	Jordan	Staal	A
16	C	Elias	Lindholm	
18	C	Jay	McClement	
20	RW	Sebastian	Aho	
21	RW	Lee	Stempniak	
33	C	Derek	Ryan	
36	C	Patrick	Brown	
42	C	Joakim	Nordstrom	
49	C	Victor	Rask	
53	C	Jeff	Skinner	A
71	C	Lucas	Wallmark	
86	LW	Teuvo	Teravainen	
30	G	Cam	Ward	
31	G	Eddie	Lack	

New York Islanders

2	D	Nick	Leddy	
3	D	Travis	Hamonic	
4	D	Dennis	Seidenberg	
14	D	Thomas	Hickey	
44	D	Calvin	de Haan	
50	D	Adam	Pelech	
12	C	Josh	Bailey	
15	RW	Cal	Clutterbuck	A
16	LW	Andrew	Ladd	A
18	C	Ryan	Strome	
24	RW	Stephen	Gionta	
25	LW	Jason	Chimera	
27	C	Anders	Lee	
29	C	Brock	Nelson	
66	C	Joshua	Ho-Sang	
72	LW	Anthony	Beauvillier	
86	LW	Nikolay	Kulemin	
91	C	John	Tavares	C
1	G	Thomas	Greiss	
30	G	Jean-Francois	Berube	

Figure 3–5: Screenshot of SPORTLOGiQ's "Click & Correct System" web application used for correcting player trajectory data and player number identity to generate ground truth data. The video frame shows bounding boxes surrounding the players and the player number label at the bottom left corner of each bounding box.

To properly train our model to identify hockey players based on the features of the jersey number, we required tracklet examples in our dataset to contain one or more frames with fully visible jersey numbers. Although the tracklets were accurately labelled, more than half of the tracklets were considered as false positives because they did not contain any frames with the player's jersey number fully visible. Usually, an experienced human annotator can accurately predict a player's jersey number using player or game context, even with the lack of visual cues of the jersey number. However, for our purpose, a sparsely annotated dataset with many false positives will not be sufficient to train our model to understand the visual features required to distinguish between different players.

We automated the process of removing the false positives with a classification step by training a CNN classifier, a 10-layer ResNet model. We used the classifier to perform the binary classification task of determining whether the jersey number is visible or not in player bounding box images. This ResNet-10 model was trained and tested on a dataset of player bounding box images, which included a subset of the SPORTLOGiQ NHL-HP dataset. The training set contains 26,000 training examples and the test set contains 3,000 test examples. This dataset was manually annotated and contains 2 classes: one class for images where the jersey number is fully visible and another class for images with no visible jersey number. After training, we tested the model on the test set, on which the model scored a classification accuracy result of 99.86%. The high score indicated that the model was able to detect the presence of a jersey number in an image. Having successfully trained the model as a binary classifier, we used the model to evaluate every image in the annotated tracklets to determine whether the jersey number was visible in any of the images in a tracklet. Tracklets with no visible jersey numbers in any of the images were identified as a false positive and removed from the NHL-HPT dataset.

Lastly, a final validation step was performed to manually check for and remove any false positives that may have slipped pass the classification step. Since the majority of false positives were already removed from the dataset during the classification step, the manual efforts required for this validation step were greatly minimized.

3.3 Visual Feature Extraction with CNNs

Convolutional Neural Networks (CNN) have progressed rapidly over the years and are now well established as a powerful model for image recognition problems.

CNNs excel at the task of learning and interpreting visual features to understand the contents of an image. Prompted by recent state-or-the-art results in image recognition, we studied their performance in extracting visual features for player identification when the most prominent visual cue available was the jersey number.

CNNs are useful in extracting and encoding spatial features into a sequence to produce a fixed-length feature vector representation of an image. In our approach, player bounding box images from a tracklet are input to the CNN layers as a visual input and the CNN generates feature vector representations for the player bounding boxes. These vector representations of the spatial encoding can then be passed into an LSTM layer as a sequential input in order for the LSTM to learn the sequential data from the input.

We trained and compared several different CNN architectures on the SPORTLOGiQ NHL-HP dataset. Since this dataset only contains bounding box images of players with the jersey number fully visible, we could evaluate how well the different CNN architectures can classify jersey numbers. The CNN architectures we studied include a standard CNN base model implemented in the Long-term Recurrent Convolutional Network (LRCN) [3], the Residual Network (ResNet) [1], and the Densely Connected Convolutional Networks (DenseNet) [2]. These architectures were described in *Chapter 2 Related Works*.

In our experiments, we tested four specific networks. The first was a standard CNN similar to AlexNet [7] and was introduced in [3] as the CNN base for the LRCN model. We also evaluated two ResNet models: one with 50 layers (ResNet-50 [1]), and the other with 10 layers (ResNet-10 [11]). Finally, we tested a DenseNet model with 121 layers (DenseNet-121 [5]). From our experimental results, as will be discussed in Chapter 4 Experiments, the ResNet-10 model demonstrated the best compromise in regards to training time and performance. Thus, we implemented the ResNet-10 into our proposed CNN+LSTM network, which will be discussed in the following section, *Chapter 3.4 Proposed Approach*.

3.4 Proposed Approach

While convolutional neural networks (CNNs) can achieve superior performance in visual image recognition tasks by automatically learning complex features, they can only operate on the spatial domain of single static images. In video recognition tasks on the other hand, additional information in the time domain is available due to a video's inherent temporal dynamics. In order to make use of the temporal component, we must be able to model sequential data of varying lengths. Given a sequential input, such as a tracklet, our goal was to capture the dynamic information and learn a global description to predict a single label for a player more accurately. In this section, we describe the architecture of our proposed ResNet+LSTM network and the late fusion method for determining a single label prediction for a player tracklet. Our approach is summarized in a flowchart in Figure 3–6.

[5] The DenseNet model we used was converted from the original Torch model Huang2016 into a Caffe format and is publicly available at https://github.com/shicai/DenseNet-Caffe.

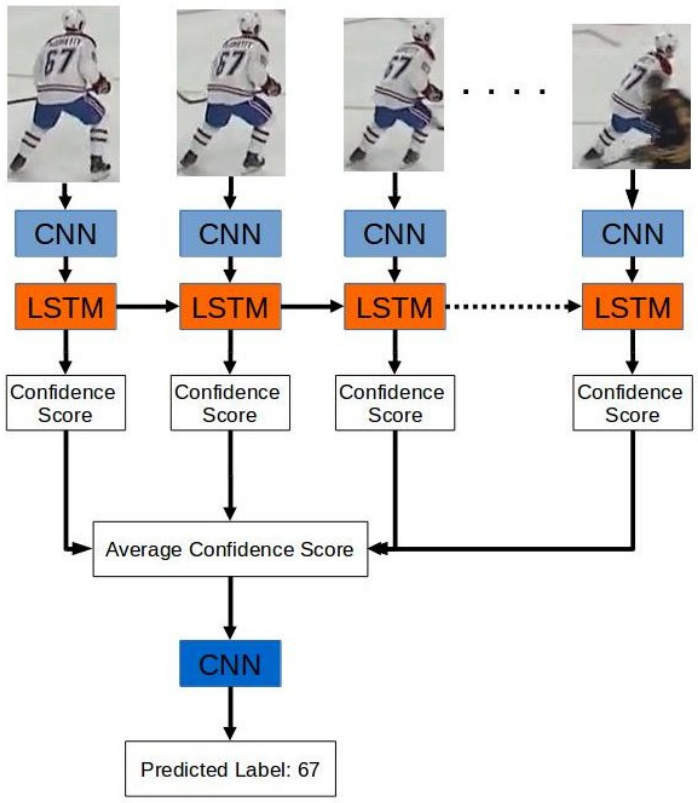


Figure 3–6: Flowchart of our proposed approach. Each frame in the tracklet feeds into the ResNet+LSTM network to generate a confidence score for that frame. The confidence scores are combined by arithmetic mean and fed into a secondary 1D-CNN classifier to output the predicted label for the tracklet.

3.4.1 ResNet+LSTM Architecture

Our proposed framework combines a CNN with LSTM units to allow sequential inputs and this work is influenced by the video activity recognition instantiation of the Long-term Recurrent Convolutional Neural Network (LRCN) [3] model. In [3], the authors train a LRCN to recognize human activity in videos from the UCF101 dataset [41]. The LRCN consists of a CNN network and a LSTM layer with 256 hidden units. As discussed in the previous section, the CNN network is similar to AlexNet [7] and the CaffeNet [9] reference model and also shares similar hyperparameters with the network used in [10].

A variable length video is treated as a sequential input by the LRCN, where each video frame is considered as a single time-step in the sequence. For a full sequence, each frame is taken in succession as a visual input and fed through the CNN layers to transform the input into a fixed-length vector representation. These vector representations are then fed into the LSTM layer where the temporal dynamics of the human action can be learnt to give each frame input a confidence score for the predicted action. To recognize the overall action in a sequence, a late-fusion method of combining the confidence scores of all the frames is used to predict a single action for the sequence. This is done by averaging the confidence scores across all frames to determine the action label with the highest probability.

In our scenario, player tracklets are sequences of player bounding box images and thus, can be treated as variable length videos, where each bounding box is viewed as an input frame. For our approach, we adopt the ResNet-10 [11] architecture, mentioned in the previous section, as the CNN base of our CNN+LSTM model, which we can now refer to as the ResNet+LSTM model. We chose to use the ResNet-10 model due to its relatively shallow architecture for faster training and runtime and high performance compared to the other CNNs we experimented with. Details on the experimental results will be discussed in *Chapter 4 Experiments*.

For the ResNet base of our ResNet+LSTM model, we keep all the hyperparameters and layers, up to the last average pooling layer, the same as the original

ResNet-10 [11]. The input frames first pass through a batch normalization layer, which eliminates the need for mean subtraction during training. The first convolutional layer feeds into a batch normalization layer, a ReLU activation function, and a max pooling layer. After that, there are four residual 2-layer stacks, which were first introduced in [1] for non-deep residual networks, as discussed in *Chapter 2 Related Works*. Each stack consists of two convolutional layers with a batch normalization layer and a ReLU activation function in between the two convolutional layers. At the end of each stack is an element-wise sum of the second convolutional layer and the input into the stack via a shortcut connection as depicted in Figure 2–1. The residual learning of each stack can be defined as:

$$y = F(x, W_i) + x \tag{3.1}$$

In Equation 3.1, x and y are the input and outputs of the stack. The function $F(x, W_i)$ describes the mapping to be learned over a series of layers. For the 2-layer stacks in our network, the function $F(x, W_i) = W2(W_1 x)$, where represents the batch normalization and ReLU function and W_1 and W_2 denotes the weights of the first and second convolutional layers. However, Equation 3.1 is only computable if the dimensions of x and F are equal. In the second, third, and fourth stack, a projection shortcut is required to match the dimensions of x with that of F. This is done by performing a linear projection W_s on the input of the stack with 1x1 convolutions. Thus, the residual learning of each stack is defined as:

$$y = W_2 \sigma(W_1 x) + W_s x \tag{3.2}$$

The stack output then feeds into another batch normalization layer and ReLU activation function. The last stack feeds into a average pooling layer, where the pooling is performed globally over each feature map to reduce its spatial size to 1x1. The output of the average pooling layer is a 512-dimensional vector representation of a frame, which feeds into a LSTM layer. The LSTM layer has 256 hidden units and receives a series of 16 512-dimensional vectors as an input sequence from the CNN layers. We apply a dropout rate of 0.5 to the LSTM output before feeding into a fully connected layer with a softmax activation function. The dropout technique sets the output of a neuron to zero with a probability of 0.5 to limit the number of neurons from activating during the backpropagation training process. The softmax classifier outputs an 81-dimensional vector (for 81 classes in our dataset) as a confidence score to predict the label for each input frame. The architecture of our ResNet+LSTM model is summarized in Table 3–1.

3.4.2 Player Tracklet Identification

The ResNet+LSTM network generates a confidence score for the predicted label of each frame in a tracklet as the output of the softmax classifier of the network. But our goal is to predict a single overall label for the entire player tracklet. Given a player tracklet T_s with N frames, we can determine a single label for that tracklet by considering all N confidence scores in the tracklet. We used a late score-level fusion method to combine all the confidence scores from different frames to maximize our label prediction result for the tracklet.

Let us denote M as the number of classes in the tracklet dataset with S tracklet examples. Each frame label confidence score is a vector $F = [p_1, p_2, ..., p_M]$, where

Input	$240 \times 320 \times 3 \rightarrow 227 \times 227 \times 3$
Conv1	7×7, 64, stride 2, pad 3
Max Pool	3×3, stride 2
Conv2_x	3×3, 64, stride 1, pad 1 3×3, 64, stride 1, pad 1
Conv3_x	3×3, 128, stride 2, pad 1 3×3, 128, stride 1, pad 1
Conv4_x	3×3, 256, stride 2, pad 1 3×3, 256, stride 1, pad 1
Conv5_x	3×3, 512, stride 2, pad 1 3×3, 512, stride 1, pad 1
Avg Pool	global pool
LSTM	256 hidden units, sequence of 16 frames
Dropout	ratio = 0.5
FC	81 classes

Table 3–1: Network architecture of the ResNet+LSTM model excluding batch normalization and ReLU activation functions. Hyperparameters including filter size, number of filters, and stride are summarized for the convolutional layers. The convolutional layers are grouped by their respective residual layer stacks.

each p_m denotes the confidence score for the mth label to be correct. For a tracklet with N frames, the ResNet+LSTM network will output N confidence scores as follows:

$$\mathrm{F}_1 = [p_1^1, ..., p_1^m, ..., p_1^M] \epsilon R^M$$

$$\vdots$$

$$\mathrm{F}_n = [p_n^1, ..., p_n^m, ..., p_n^M] \epsilon R^M$$

$$\vdots$$

$$\mathrm{F}_N = [p_N^1, ..., p_N^m, ..., p_N^M] \epsilon R^M$$

We can concatenate all N confidence score vectors to produce a NxM confidence score matrix for each tracklet. A confidence score matrix is an ideal way to organize the data to input into the late score-level fusion method, as detailed in the next section.

3.4.3 Late Fusion Scheme

Late score-level fusion methods combine confidence scores from different classifiers to improve upon the performance of an individual classifier. These confidence scores can come from multiple sources that are evaluated by different classifiers trained with unique features. Although we applied the same ResNet+LSTM classifier on multiple sources (i.e., bounding box frames), we aim to combine the confidence scores from every frame in a tracklet to achieve better results in labeling tracklets. Since late fusion methods require confidence scores from the entire tracklet before predicting the final label, they are only suitable for off-line analysis. If we wish to do player identification in real-time, other techniques will need to be explored.

In our late fusion approach, we first combined all the confidence scores of a tracklet from the ResNet+LSTM classifier using the arithmetic mean. The average confidence score for each tracklet is calculated by averaging the probabilities for each label across all the frames (or averaging the columns of the confidence score matrix). Since there are 81 different class labels, the average confidence score is an 81-dimensional vector given as $F_{avg} = [p_1, p_2, ..., p_M] \epsilon R^M$, where $M = 81$. Instead of viewing the class with the maximum likelihood in F_{avg} as the predicted label, we can predict the label by providing F_{avg} as a feature vector input to a secondary supervised classifier. This secondary classifier is a shallow 1-dimensional convolutional neural network (1D-CNN) with 6 convolutional layers and 3 fully connected layers in total. Some of the convolutional layers are arranged such that there are 2 residual stacks within the network. A shortcut connection is formed between the outputs of the second and third convolutional layers to perform element-wise addition, as well as between the fourth and fifth layers. The first 2 fully connected layers both have 256 neurons and the last fully connected layer has 81 to serve as the softmax classifier. The architecture of the 1D-CNN is summarized in Table 3–2.

Input	1×81 vector
Conv1	1×3, 20, stride 1, pad 1
Conv2_1	1×3, 50, stride 1, pad 1
Conv2_2	1×3, 50, stride 1, pad 1
Eltwise Sum	Conv2_1+Conv2_2
Conv3_1	1×3, 70, stride 1, pad 1
Conv3_2	1×3, 70, stride 1, pad 1
Eltwise Sum	Conv3_1+Conv3_2
Conv4	1×2, 70, stride 1, pad 1
Max Pool	1×2, stride 2
FC1	256 hidden units
ReLU	$\downarrow$
FC2	256 hidden units
ReLU	$\downarrow$
FC3	81 classes

Table 3–2: Complete network architecture of the 1D-CNN classifier

Given F_{avg} as a feature vector input, the 1D-CNN can detect the patterns within F_{avg} and match them to the patterns that commonly occur in the average confidence scores of the corresponding label. For example, the average confidence score for a tracklet labeled as "45" as ground truth may tend to have higher probabilities for jersey number labels with the digits "4" and "5", such as "45", "43", or "15". This is usually due to the inaccuracies of the player bounding box classification process or occlusion of one of the digits in the frames. Taking advantage of these patterns in the confidence scores has proved to increase the accuracy results by 4.22%, as will be discussed in *Chapter 4 Experiments*.

3.5 Training Methodology

3.5.1 CNN Training

Our ResNet+LSTM network's training process involved pre-training the CNN section first before training the entire network from end to end. All of our model training used backpropagation to update the network parameters. For the CNN section, we used the pre-trained Caffe model of ResNet-10 [11] that was made publicly available. The model was pre-trained on the large ImageNet [62] dataset, which contains 1.2 million images with 1000 categories. This supervised pre-training results in a strong weight initialization for a more efficient convergence and better generalization. As demonstrated in [10], a model pre-trained on ImageNet can achieve a significant improvement in classification accuracy in other datasets over a non-pretrained model.

To train the CNN layers of our network, we first pre-trained the ResNet-10 [11] model on a subset of the Street View House Number (SVHN) [63] dataset. The SVHN [63] dataset is a collection of cropped images of house numbers from Google Street View images. Our subset of the SVHN dataset contains 16,222 examples that are categorized by 10 classes, where each class is associated with a potential digit. By pre-training on the SVHN dataset, we can further improve the weight initialization and generalization of the ResNet-10 for the task of identifying jersey numbers, which was verified by our experiments in Chapter 4 Experiments. Once pre-trained, we fine-tuned the ResNet-10 on the player bounding box images in the SPORTLOGiQ

NHL-HP dataset. We allowed all the layers in the network to be fine-tuned, since we had a sufficient number of data examples in the SPORTLOGiQ NHL-HP dataset to prevent overfitting.

3.5.2 ResNet+LSTM Training

When we trained the ResNet+LSTM network from end to end, we borrowed the weight parameters from the fine-tuned ResNet-10 to initialize the weights of the CNN layers in the ResNet+LSTM network. The network trained on inputs of short sequences of 16 consecutive bounding box frames (about a half of a second). Similar to the training method used in [3] to mimic data augmentation, each training input sequence of 16 consecutive frames was randomly selected from a tracklet T_s with Nframes. This input process created a higher number of possible training inputs for the network. Each tracklet T_s in the training set can provide $N-15$ different training input sequences of 16 consecutive frames. The frames were resized to 320×240 and a random 224×224 crop was taken from each frame. During the end-to-end training, we froze the CNN layers by setting the learning rate of the CNN layers to 0 to prevent further updates to the weight parameters taken from the fine-tuned ResNet-10. In our experiments, we found this step significantly improved the performance of our network.

3.5.3 1D-CNN Training

For our late fusion approach, we used a shallow 1-dimensional CNN (1D-CNN) as the secondary classifier to predict the overall label for a tracklet, given the tracklet's confidence score as an input. To train the 1D-CNN, we created a dataset of tracklet confidence scores by collecting the outputs from the ResNet+LSTM network. A training set was created by using the training tracklet examples from the NHL-HPT dataset as inputs to the network and A test set was created by using the test tracklet examples as inputs. Since we were inputting training tracklet examples into the ResNet+LSTM network that were previously used to train the network itself, there could be some concerns regarding the confidence score outputs of the network. This was due to the possibility that the network could overfit on the training data. Overfitting occurs when the model starts to memorize the training data rather than learn to generalize on the patterns in the data. As a result, the model will achieve a high accuracy on the training data it has already been exposed to, but do poorly on any unseen data. In our case, what this can mean is that the confidence scores of the training examples produced by the network may not be an accurate representation of the confidence scores in the test set to properly train the 1D-CNN. However, in our experiments, the network did not noticeably overfit on the training examples as the training and testing accuracies were similar. This was perhaps due to ResNet+LSTM network's training process, where short random clips of 16 frames were taken from a tracklet as an input during each training iteration. The randomization of the training process helped to prevent overfitting by increasing the variation of our training dataset. The ResNet+LSTM network's ability to generalize well with the tracklet training dataset allowed us to use the confidence scores from the training examples as appropriate data to train the 1D-CNN. Since the 1D-CNN is fairly shallow, training time only took about 20 minutes to run 195 epochs.

3.6 Summary

In this chapter, we gave a comprehensive description of the entire methodology. We established that the task of player identification using player tracklet data was similar to video classification problems. Thus, we proposed an approach that required a convolutional and recurrent neural network in order to learn the temporal data from an input of a sequence of images. We used the SPORTLOGiQ NHL Hockey Player (NHL-HP) dataset to train and test several CNN architectures and chose the ResNet-10 as the base of our ResNet+LSTM network. We also created a new NHL Hockey Player Tracklet (NHL-HPT) dataset to train and evaluate the ResNet+LSTM network. As a last step in our proposed approach we used a late score-level fusion method to predict a single label for a tracklet. We combined all the confidence score outputs of the network by the arithmetic mean and input the mean score into a secondary 1-dimensional CNN classifier to generate the tracklet's overall label.

In the next chapter, *Chapter 4 Experiments*, we will discuss the results from the experiments we conducted in detail. We will explain the comparisons we made for different architectures and review the process we went through to improve our model's tracklet classification accuracy.

CHAPTER 4
Experiments

In this chapter, we present the results of our experiments. We test the CNN models and ResNet+LSTM network that were previously described in Chapter 3 Methodology on the SPORTLOGiQ NHL Hockey Player (NHL-HP) dataset and the NHL Hockey Player Tracklet (NHL-HPT) dataset, respectively. We will first compare the results of the experiments on the CNN models and elaborate on how we chose to adopt the ResNet-10 [11] as the CNN backbone of our ResNet+LSTM network. Following that, we experiment with performing player identification using player tracklets with the ResNet+LSTM network and test various late score-level fusion techniques to improve upon the network's performance. All experiments were implemented in the Caffe [1] framework and executed on a Tesla K40 GPU.

4.1 CNN Comparison

In this section, we compare the performances of each of the four CNN models to determine the appropriate CNN model to use as the backbone of our CNN+LSTM network. We investigate their ability to classify a player based on the visual features of a jersey number when given a single image input. As we were only dealing with

[1] The implementation of Caffe we used was updated to support recurrent networks with LSTM units and is available at https://github.com/BVLC/caffe

single images, we considered this exercise as a single image classification task. In this set of experiments, we also wanted to observe the effects of pre-training on the CNNs' performance. Thus, we implemented two different training procedures for each model: one that included pre-training, and another that did not. We then evaluated the CNNs on the SPORTLOGiQ NHL-HP dataset and chose the most suitable CNN to act as the hierarchical feature extractor for our CNN+LSTM network based on the results.

4.1.1 Implementation Details

To compare the performance of the different CNN models and examine the effects of pre-training, we implemented two training procedures. In the first training procedure, we trained each CNN model directly on the SPORTLOGiQ NHL-HP dataset. Since the SPORTLOGiQ NHL-HP dataset is a private dataset, it is publicly unavailable. In the second training procedure, we first pre-train each model on the Street View House Number (SVHN) [63] dataset before fine-tuning on the SPORTLOGiQ NHL-HP dataset. By comparing the results of the two training procedures, we can evaluate the effectiveness of the additional pre-training from the street number images. If the results from pre-training are favorable, it will show that the features learnt from street numbers can be transferred to adapt to features of player jersey numbers. Due to time constraints, we did not rigorously perform hyperparameter tuning and used a learning rate of 0.001 and momentum of 0.9 for all of the CNN training. Note that all of the CNN models used were already pre-trained on the ImageNet [62] dataset and are publicly available online [2] .

In the first training procedure, we trained each CNN model directly on the 45,468 training images from the SPORTLOGiQ NHL-HP dataset and evaluated the model on the 3,885 test images from the same dataset. As mentioned in Chapter 3 Methodology, this train/test split was done randomly. To maintain a consistency across all our experiments, we trained each model for approximately 140 epochs on the Tesla K40 GPU.

During the training process, the weights of the networks are constantly updated at each iteration. We wished to identify the moment when the weights become most optimized for the current task without overfitting to the training set. We did this by analyzing the loss curve plot of both the training and test phases. Generally, when training neural networks, the training loss will ideally decrease and converge to a value close to zero over time. The test loss will also decrease over time, but after a certain point, the test loss may begin to increase. This is an indicator that the network was trained for too long and started to overfit on the training set and no longer able to generalize on the test set. Thus, the iteration with the most optimal weights occurs when the test loss converges to its minimum. During the training process, we saved a "snapshot" of the network's parameters after every 5,000 training

[2] The standard CNN, ResNet-10, ResNet-50, and DenseNet-121 models are publicly available at https://github.com/LisaAnne/lisa-caffe-public, https://github.com/cvjena/cnn-models, https://github.com/KaimingHe/deep-residual-networks, and https://github.com/liuzhuang13/DenseNet respectively

iterations and kept the one that corresponded to the minimum of the test loss curve to obtain the optimized parameters of the network. In Table 4–1, we show the batch size values and number of iterations we used to train each CNN model on the SPORTLOGiQ NHL-HP dataset.

CNN Model	Pre-training on SVHN (for 197 epochs)		Training/Fine-tuning on NHL-HP (for 140 epochs)	
	Batch Size	Iterations	Batch Size	Iterations
Standard CNN [3]	128	25,000	128	50,000
ResNet-10 [11]	128	25,000	128	50,000
ResNet-50 [1]	16	200,000	16	400,000
DenseNet-121 [2]	16	200,000	16	400,000

Table 4–1: The above table shows the training variables we set for our training instances, including batch sizes and the number of iterations. The breakdown for training/fine-tuning on NHL-HP is the same for the two training procedures: one with and one without pre-training on SVHN.

Our second training procedure included a pre-training step before following the same process as the first training procedure above for fine-tuning the model. In the pre-training step of the process, we pre-trained the CNN models on a subset of 16,222 images from the Street View House Number (SVHN) [63] dataset for approximately 197 epochs. After all the training iterations were complete, we plotted the loss curve of both the training and test phases. In the case of the ResNet-10 model, we observed in Figure 4–1 (bottom left plot) that the test loss curve was at a minimum after 15,000 iterations. Thus we took the "snapshot" of the model at 15,000 iterations as our optimized pre-trained ResNet-10 model. In the fine-tuning step, we fine-tuned

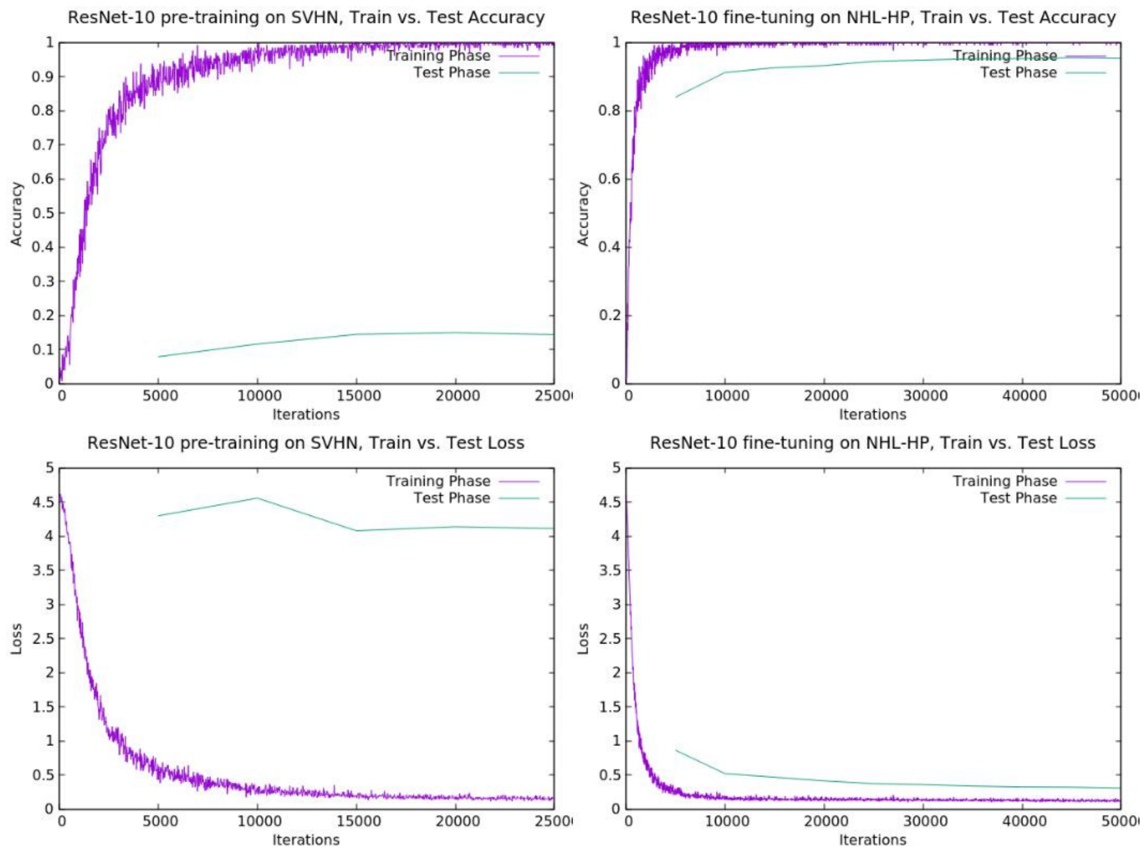


Figure 4–1: The ResNet-10's loss curve plots (bottom two plots) and accuracy curve plots (top two plots) of the training and test phases during pre-training on SVHN dataset and fine-tuning on the SPORTLOGiQ NHL-HP dataset are shown above. Note that the accuracy is much lower and the loss is much higher during the pre-training step (top left and bottom left plots) since the network was evaluated on the SPORTLOGiQ NHL-HP dataset instead of the SVHN dataset.

the pre-trained models on hockey player images from the SPORTLOGiQ NHL-HP dataset for 140 epochs by adopting the same training variables as the first training procedure. During the fine-tuning process, we set the learning rate multiplier of each layer in the network to 1 to allow the weights of all layers to be updated at every iteration. Similar to the first step, we found our optimized fine-tuned model by checking the test loss curve (Figure 4–1, bottom right plot) for its minimum, which was after 50,000 iterations for the ResNet-10. The batch size and iteration variables used for this training procedure are summarized in Table 4–1.

4.1.2 Results and Discussion

In Table 4–2, we present the test accuracy and training duration of the different CNN models on the SPORTLOGiQ NHL-HP dataset. For the standard CNN [3] and ResNet-10 [11], pre-training on the SVHN dataset provided a boost of 8.69% and 2.74% in performance, respectively. However, for the ResNet-50 [1] and DenseNet-121 [2], the pre-training on the SVHN dataset did not offer any improvements. Instead, the accuracy performance dropped by 2.21% and 0.47% for the ResNet-50 and DenseNet-121 respectively.

Let us first discuss the mechanics of pre-training, which is a form of transfer learning. Typically, when training a neural network on a dataset to perform an image classification task, the training can be started by randomly initializing the weights. Then throughout each iteration of the training process, the weights are optimized to minimize the loss function. Once the training results are satisfactory, the most optimized weights can be saved as the final parameters of the network. Suppose we wish to train a second neural network to perform another image classification task on a different, but similar dataset. Instead of starting the training with a random weight initialization as before, which may not be the most efficient method of weight initialization, we can transfer the weights from the first network over to the second network to initialize it. In this scenario, the first network is referred to as a pre-trained network and we are fine-tuning the pre-trained network as the second network to perform another task. By transferring the weights from a pre-trained network, we can reuse weights that were already optimized for a similar task. This process is known as transfer learning and was shown in [64] to deliver a stronger weight initialization for a network to achieve a more efficient convergence and a boost in generalization.

The correlation noted in the test accuracy results was that pre-training on the SVHN dataset allowed the shallower CNNs (standard CNN and ResNet-10) to generalize better to the SPORTLOGiQ NHL-HP dataset. However, the deeper CNNs (ResNet-50 and DenseNet-121) did not receive any benefits from the pre-training and actually observed a drop in performance compared to the instance when the weights were randomly initialized. Unexpectedly, the shallower ResNet-10 achieved higher performance than its deeper counterpart, the ResNet-50. This may be due to overfitting by the deeper network and the problem may have been amplified by the class imbalance of the dataset. From this experiment, we can conclude that while transfer learning can improve the generalization of relatively shallow CNNs, it can also undermine the generalization of deeper CNNs with a higher number of layers. As CNNs increase in depth, the higher layers develop high-level features that are increasingly more fitted to the specific dataset it was trained on. While lower layers in any network still have highly generalized low-level features, the features become less generalized in higher layers. As a result, the weights in deep CNNs are no longer transferable because the transferred weights are no better at generalizing to a target dataset during training than randomly initialized weights.

We also observed the training duration for each of the CNNs. As expected, the time required to train for 140 epochs by the deeper networks was much greater than that of the shallower networks, which only took over 18 hours. Networks with a greater number of layers are less memory-efficient since training using backpropagation requires memory storage for all the layers' parameters and outputs. The DenseNet is particularly memory intensive due to the large number of connections between layers in the network, which resulted in requiring the highest computation time out of all the CNNs we tested. On a Tesla K40 GPU, the batch size to train the DenseNet-121 and ResNet-50 was limited to 16 due to hardware constraints. On the other hand, we were able to train the standard CNN and ResNet-10 with a batch size of 128 using the same GPU.

In the end, we chose the ResNet-10 as the CNN backbone of our CNN+LSTM network. When we pre-trained the ResNet-10 on the SVHN dataset, it delivered the second best performance with a 95.34% accuracy rate. The DenseNet-121 achieved the highest accuracy rate of 97.43%, but at the expense of a much longer training time. A base CNN with a lower training duration can allow us the freedom to retrain the CNN+LSTM network to perform multiple trial and error experiments with different hyperparameters. With the ResNet-10, we achieved a high performance with an acceptable training time. Thus, from our experiments, the ResNet-10 proved

CNN Model	Training Duration on NHL-HP (for 140 Epochs)	Test Accuracy on NHL-HP	
		Pre-trained on SVHN	No pre-training on SVHN
Standard CNN [3]	18.45hrs	84.24%	75.55%
ResNet-10 [11]	18.82hrs	95.34%	92.6%
ResNet-50 [1]	64.35hrs	86.38%	88.59%
DenseNet-121 [2]	142.87hrs	96.96%	97.43%

Table 4–2: Comparison of different CNN models' training duration and their test accuracy performance for the two different training procedures. Test accuracy results are based on their performance on the SPORTLOGiQ NHL-HP test dataset.

to be the best CNN option to adopt in our CNN+LSTM network. In the next section, we will discuss how we implemented the ResNet-10 with a LSTM layer to perform player tracklet identification.

4.2 Player Tracklet Identification

After our experiments that involved single image classification, we moved on to the main purpose of this research, which is to classify player tracklets with a single label. In the previous section, we performed single image classification to determine the player in a single image with the confidence score that the CNN produced. However, for player tracklet identification, we have a sequential set of images for each tracklet. To identify a player tracklet, we needed to first analyze and generate a confidence score for each image in the tracklet before combining all the confidence scores to calculate the overall score of the tracklet. We then experimented with an additional step of classifying the combined confidence score with late score-level fusion methods to produce the overall label of the tracklet with higher accuracy.

In this set of experiments, we evaluated several models on the NHL Hockey Player Tracklet (NHL-HPT) dataset. We compared the performance of our proposed ResNet+LSTM network with two baseline models: the ResNet-10 [11] network and the LRCN [3] model. Since the architecture of the ResNet+LSTM network contains some similarities with that of the baseline models, we observed how the architecture changes reflected in the ResNet+LSTM network improved the network's performance compared to the baselines in these experiments.

4.2.1 Implementation Details

We compare our proposed ResNet+LSTM network with the LRCN [3] and the ResNet-10 [11] models. Our ResNet+LSTM is architecturally similar to the LRCN as both networks combine the use of a CNN as a hierarchical feature extractor and a LSTM layer as a recurrent sequence learning module. Whereas the LRCN uses a standard CNN as the network's hierarchical feature extractor, the ResNet+LSTM incorporates a ResNet-10 model as the hierarchical feature extractor. Consequently, the ResNet+LSTM is also architecturally similar to the ResNet-10 model if we discount the addition of a LSTM layer. From our experimental results, we will determine if we obtained a boost in performance with the ResNet+LSTM network due to the architectural changes.

We evaluated the two baseline models and the ResNet+LSTM network on the same test set from the NHL-HPT dataset and compared their results. In all our experiments, we used each model to output the confidence score of each frame in a tracklet and averaged all the confidence scores to calculate the overall score for the tracklet. Later on in Chapter 4.3 Late Fusion Methods, we discuss alternative methods we tested in place of averaging to calculate the tracklet's overall score. Below are the implementation details for the three models:

ResNet-10. Since the CNN base of our proposed ResNet+LSTM architecture is the ResNet-10, we compared the results of our ResNet+LSTM network with just the base to demonstrate the improvements we gained by adding a LSTM layer. The ResNet-10 model we used in this experiment was trained on single images in the SPORTLOGiQ NHL-HP dataset as implemented earlier in the Chapter 4.1 CNN Comparison section. To classify tracklets with the ResNet-10, we input each frame from a tracklet into the network individually to produce a confidence score for each one. Note that unlike a network with a recurrent module, such as the LRCN and ResNet+LSTM, a CNN can only consider data from the current input and cannot use data from the previous or next input to influence the classification task. So the ResNet-10 can only classify one frame at a time without data from other frames. Once all the frames in the tracklet were evaluated by the network, the confidence scores of all the frames were averaged to calculate the overall score of the tracklet. As it will be discussed later in the *Chapter 4.3.2 Results and Discussion* section, taking the average is shown to work well in our case compared to all other methods we tried.

LRCN. Since our ResNet+LSTM network was largely inspired by the LRCN [3] model, we evaluated the LRCN on our NHL-HPT dataset to serve as a performance benchmark for comparison with the ResNet+LSTM. We followed a training methodology similar to the one in Donahue2016 to train and test the LRCN on our NHL-HPT dataset. Earlier, in the *Chapter 4.1 CNN Comparison* section, we already trained a standard CNN, which is the base of the LRCN. We transferred the weights of the standard CNN to the LRCN before we started training with input sequences of 16 consecutive frames randomly selected from a tracklet from the NHL-HPT dataset. We trained the LRCN for 100,000 iterations and used the same hyperparameters as the authors in [3]. For each training iteration, we used a batch size of 24 16-frame input sequences. We set up the training phase to save a "snapshot" of the model after every 5,000 training iterations to preserve the values of the weights at evenly spaced intervals. When the training phase was completed, we ended up with 20 model "snapshots", each containing a different set of weights of the network. To test which model "snapshot" has the most optimized weights for the tracklet identification task, we ran a Python script to evaluate the full NHL-HPT test set using the weights from each model "snapshot". The script allowed one tracklet input into the LRCN at a time. Since the LSTM module was set up to accept 16-frame input sequences, the LRCN processed each tracklet input by evaluating 16 frames at a time. When the LRCN has generated all the confidence scores for every frame in the tracklet, all the scores were averaged to produce the tracklet's overall confidence score. We compared the accuracy results of each "snapshot" and the weights of the "snapshot" that delivered the highest classification accuracy was chosen as the final optimized weights of the LRCN.

ResNet+LSTM. The training procedure is similar to that of the LRCN as described above. We transferred the fine-tuned weights from the ResNet-10, as described earlier in the *Chapter 4.1 CNN Comparison* section, to the CNN layers in the ResNet+LSTM network. When we trained the ResNet+LSTM end-to-end on the NHL-HPT dataset, we set the learning rate on the ResNet layers to 0 to prevent further updates to the weights. In our experiments, we also performed hyperparameter tuning on the initial learning rate (results are shown in Table 4) by training the model several different times, each with an incremental change to the initial learning rate. Similar to the LRCN experiments, each trial was performed for 100,000 iterations, where each iteration has an input of 24 16-frames sequences. We also saved a "snapshot" of the model after every 5,000 training iterations and determined the most optimized model "snapshot" with the same method we used for the LRCN, as described above.

4.2.2 Results and Discussion

ResNet+LSTM vs. Baselines

In Table 4–3, we present the test accuracy of the ResNet+LSTM, ResNet-10, and the LRCN models on the NHL-HPT dataset. Our ResNet+LSTM delivered the highest performance in player tracklet identification with an 81.22% accuracy rate, followed by the LRCN with an accuracy of 80.83% and the ResNet-10 with an accuracy of 59.08%.

Models	**Test Accuracy on NHL-HPT**
ResNet-10	59.08%
LRCN	80.83%
ResNet+LSTM	81.22%

Table 4–3: Accuracy results are based on their performance on the NHL-HPT dataset.

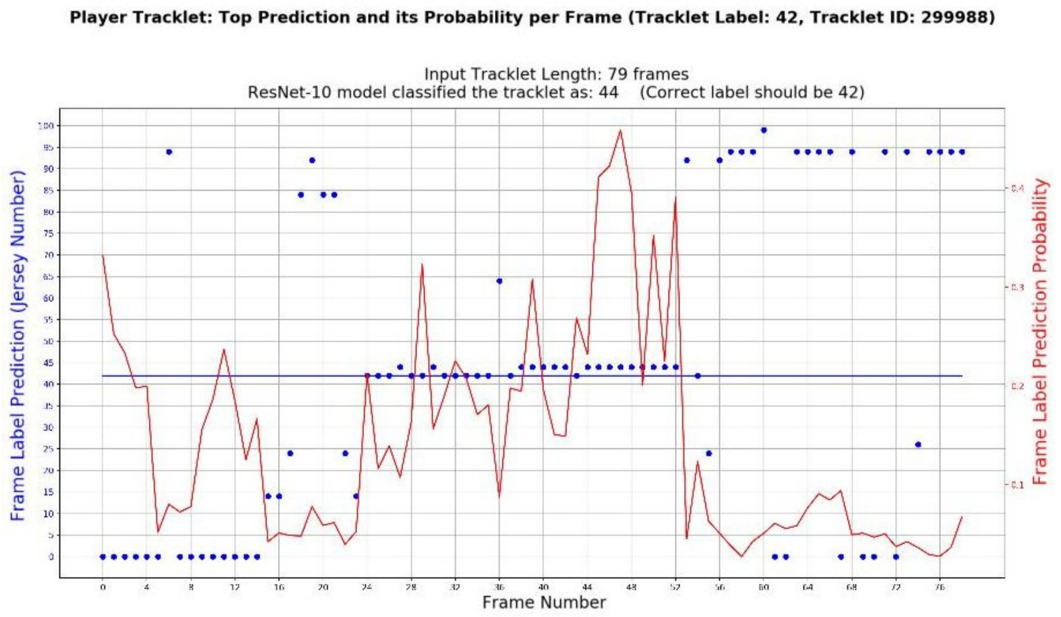
Compared to a typical CNN like the ResNet-10, the networks with LSTM layers saw a significant boost in performance by an increase in accuracy of 22.14% for the

ResNet+LSTM and 21.75% for the LRCN. The ResNet+LSTM and LRCN achieved similar results, with the ResNet+LSTM achieving a slightly better performance, due to an improved CNN base.

For a more qualitative analysis of the difference an additional LSTM layer can make on the performance of a network, we plotted the ResNet-10 and ResNet+LSTM networks' outputs on a frame-by-frame basis to observe their behavior. Figure 4–2 shows the results of the same tracklet example evaluated by both the ResNet-10 (Figure 4–2, top plot) and ResNet+LSTM (Figure 4–2, bottom plot) to compare the results between a network with a recurrent module and one without. In the two plots, the x-axis represents the frame sequence of the tracklet in chronological order. The blue dots indicate the jersey number label prediction for each frame along the x-axis, and the red curve shows the confidence probability of each label prediction. Since each frame in the ResNet-10 plot was evaluated individually, the confidence probabilities of each label prediction are independent of each other. The blue line in both plots marks the ground truth label of the tracklet, which is '42'.

We can see in the ResNet-10 plot (Figure 4–2, top plot) that the network was able to predict the correct label for several frames in the tracklet. However, the network also incorrectly classified several frames as the label '44' with relatively higher confidence probabilities. Due to the high confidence in the wrongly classified labels, the overall tracklet was incorrectly labeled as '44' instead of the true label, '42'.

In the ResNet+LSTM plot (Figure 4–2, bottom plot), significantly more frames were classified with the correct label than the frames in the ResNet-10 plot. The



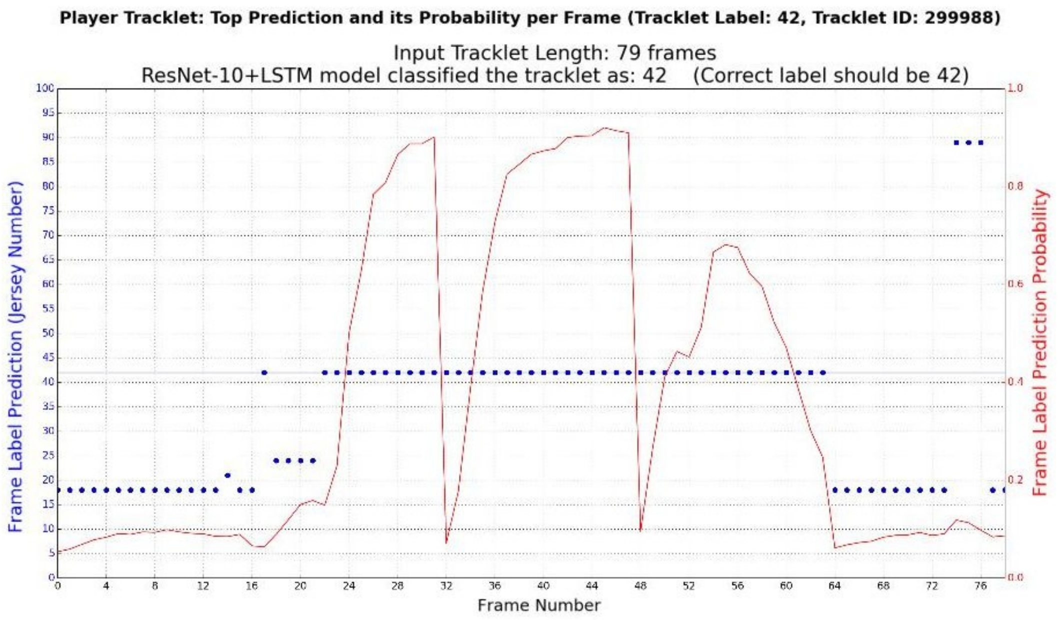


Figure 4–2: The two plots show the label prediction (blue dots) and corresponding confidence probability (red curve) for each frame in a selected tracklet example (tracklet ID: 299988). This tracklet is labeled by ground truth as '42' and contains 79 frames. For comparison, the top plot shows the label and confidence outputs predicted by the ResNet-10 and the bottom plot shows the label and confidence outputs predicted by the ResNet+LSTM model. The blue line in both plots indicates the ground truth label of the tracklet, which is '42'. Please refer to Figure 4–3 for the actual bounding box frames of this tracklet example.

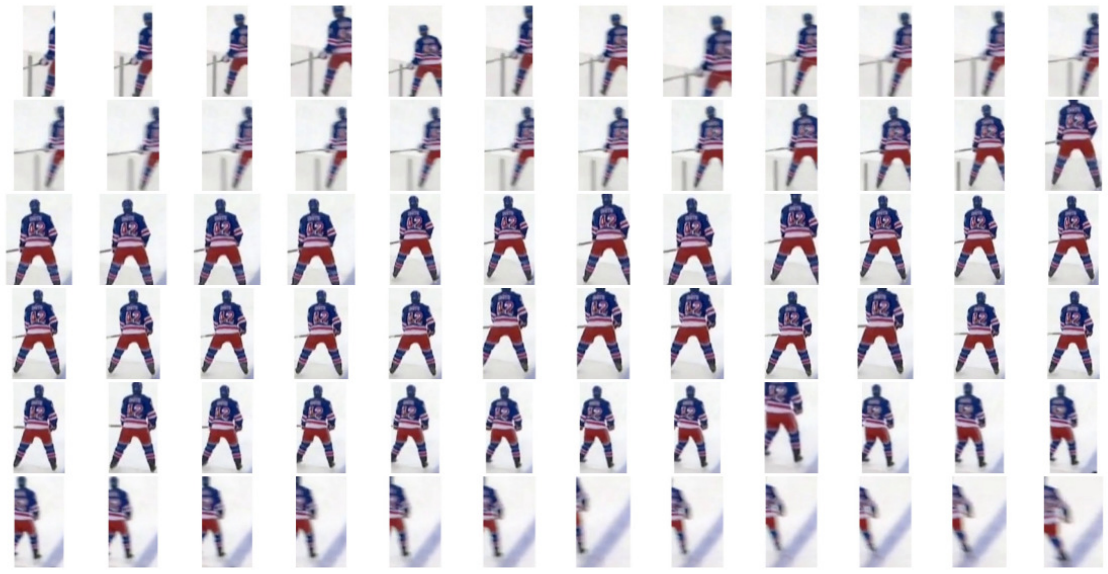


Figure 4–3: A tracklet example labeled by ground truth as “42”.

confidence probabilities of the predicted labels in the ResNet+LSTM plot were also consistently higher for labels that were correctly predicted and lower for labels that were incorrectly predicted. We also noticed in the plot that the confidence probability increases within intervals of 16 frames, such as from the 16th frame to the 31st frame and from the 32nd frame to the 47th frame. Since the LSTM layer operates on input sequences of 16 frames, it can only evaluate sections of 16 frames in a tracklet. Input sequences of greater lengths may enable better performances since a greater length will allow the LSTM to observe more frames and retrieve more relevant data. However, the constraint in sequence length was placed on the LSTM layer to accommodate the available data because all the tracklets in the NHL-HPT dataset are at least half a second (or 16 frames) or greater in length. At the beginning of each section of 16 frames, the confidence probability is low since the LSTM has not yet observed any previous frames. But as the network evaluates more frames in the section, the LSTM can accumulate information from previous frames in its current hidden state to make a label prediction for the current frame with a higher confidence. This results in the steep confidence probability increase within intervals of 16 frames in the plot. By considering observations from previous frames with the LSTM layer, the ResNet+LSTM was able to accurately predict more correct labels than the ResNet-10 for this particular tracklet example.

Hyperparameter Tuning

From our experiments, we concluded that the addition of a LSTM component can enhance the performance of a CNN model. When we determined that the ResNet+LSTM delivered the highest performance, we aimed to further optimize the model by tuning the initial learning rate hyperparameter by trial and error. As shown in Table 4–4, we tested a range of learning rates higher and lower than 0.001, which we first used to train the network. We found that an initial learning rate of 0.005 boosted the accuracy of the ResNet+LSTM on the NHL-HPT test set to 82.79% from the original accuracy score of 81.22% when the initial learning rate was 0.001.

4.3 Late Fusion Methods

Up until now, we have been using the arithmetic mean to combine the confidence scores of all the frames in a tracklet to find the overall label for the tracklet. We aimed to improve the performance of our approach by exploring other means of determining the overall tracklet label when given the frames' confidence scores as

Initial Learning rate	**ResNet+LSTM Test Accuracy**
0.0005	76.60%
0.0006	78.17%
0.0008	79.58%
0.001	81.22%
0.0012	81.77%
0.0014	81.77%
0.002	82.47%
0.003	82.71%
0.004	82.63%
0.005	**82.79%**
0.006	82.55%

Table 4–4: Learning rate comparison table for ResNet+LSTM. Accuracy results are based on their performance on the NHL-HPT dataset.

the outputs of the ResNet+LSTM network. We experimented with different late score-level fusion techniques to observe any changes in performance compared to just taking the arithmetic mean.

4.3.1 Implementation Details

To test each late fusion method, we began by concatenating all N confidence score vectors (of M dimensions) from N frames to produce an NxM confidence score matrix for a given tracklet, as described in *Chapter 3.4.2 Player Tracklet Identification.* Below is a list of fusion strategies we performed on the confidence score matrix to produce a single overall confidence score vector:

1. **Average rule** - Average all N probabilities of each individual class.
2. **Median rule** - Find the median of all N probabilities of each individual class.
3. **Sum rule** - Sum all N probabilities of each individual class.
4. **Product rule** - Multiply all N probabilities of each individual class.

5. **Logarithmic sum rule** - Sum all N log-likelihoods of each individual class.
6. **Geometric mean +** - Find the geometric mean by multiplying all N probabilities of each class and calculate the Nth root of the product. Then renormalize.
7. **Average top 16 scores** - Select the top 16 frames with the highest confidence probabilities assigned to their predicted labels and average the 16 confidence scores.
8. **Max score** - Select the frame with the highest confidence probability and assign its frame label as the overall tracklet label.
9. **Sum top X scores** - Select the top X number of frames with the highest confidence probabilities assigned to their predicted labels and sum all X confidence scores. For X, we used 16, 20, 25, 32, and 48.
10. **Sum top $Y\%$ scores** - Select the top $Y\%$ of the frames with the highest confidence probabilities assigned to their predicted labels and sum all confidence scores from the top $Y\%$ of the frames. For Y, we used 10, 20, and 30.
11. **Eliminate all "unknown" predictions** - Eliminate all frames labeled as "unknown" and take the average of remaining confidence scores to calculate overall score.
12. **Set minimum threshold on scores** - Put a minimum threshold on all confidence probabilities and set all confidence probabilities under the threshold to zero. Take the average of confidence scores with applied threshold to calculate overall score.

13. **1D-CNN** - Average the confidence scores of all the frames in a tracklet and classify the averaged confidence score with a one-dimensional CNN to produce a single label.

We start off with the first five methods (methods 1-6) that use simple mathematical operations to combine confidence scores. We also try other methods (methods 7-11) that are more selective about which confidence scores to consider to calculate the overall confidence score. This lets us take advantage of frame labels with relatively higher scores to increase our chances of predicting the correct tracklet label. We also try to adjust the confidence scores by setting a minimum threshold (method 12) to reduce interference from low confidence scores. The last method (method 13) involves treating the average confidence score of a tracklet as a feature vector to be classified by a secondary one-dimensional CNN.

We experimented the most with the 1D-CNN classifier as there were various ways to alter the 1D-CNN's architecture to influence its performance. Our input data into the classifier are relatively small spatially since they are only 81-dimensional confidence score vectors (due to the 81 classes in the NHL-HPT dataset). To design a 1D-CNN classifier for our data, we set up a shallow network based on the simple LeNet [65] architecture. Like most CNN models, LeNet was designed to operate on two-dimensional images, but our data are one-dimensional. Thus, we modified the LeNet model into a one-dimensional CNN capable of accepting vector inputs by giving all the filters a height of one and setting the vertical stride of the filters to one. We experimented with changing the network's architectural configurations and explored different hyperparameters, including filter sizes and the number of filters.

With inputs of 81-dimensional vectors, the input dimensions are relatively small so the dimensions of the data should be maintained as much as possible throughout the network. We maintained the dimensions by using filter sizes of 1x3 and used a stride of one and padding of one on each side of the feature maps. Originally, there were intermediate pooling layers and one single fully connected layer with 500 units in the LeNet. We removed the intermediate pooling layers and replaced the single fully connected layer with two fully connected layers with 256 units each. We also structured the convolutional layers into residual stacks to boost the network's performance. The design of the final 1D-CNN architecture was described in further detail in *Chapter 3.4.3 Late Fusion Scheme*.

4.3.2 Results and Discussion

In Table 4–5, we can compare the performances of the different late fusion methods we experimented with against the average rule, which we have been using to determine the test accuracy up until now. In general, the strategies that involved simple mathematical operations like the median rule, product rule, logarithmic sum, and geometric mean achieved poor results and fared worse than the average rule. The only exception was the sum rule, which achieved the same accuracy as the average rule. An issue with these simple mathematical operations was that they take into account every frame's confidence score in a tracklet without considering the difference of each frame's contribution. This is a significant factor since the frames that were assigned with a low confidence probability tend to be incorrectly classified. Without differentiating between the values of each frame's contribution, we allowed frames with a higher confidence probability to contribute equally to the

Late Fusion Methods	**Test Accuracy**
Average rule	82.79%
Median rule	64.24%
Sum rule	82.79%
Product rule	74.57%
Logarithmic sum	77.07%
Geometric mean + renormalize	78.79%
Average scores of top 16 frames	83.80%
Max score	83.02%
Sum highest scores of top 16 frames	75.51%
Sum highest scores of top 20 frames	83.96%
Sum highest scores of top 25 frames	84.27%
Sum highest scores of top 32 frames	84.19%
Sum highest scores of top 48 frames	84.19%
Sum highest scores of top 10% of frames	82.16%
Sum highest scores of top 20% of frames	84.04%
Sum highest scores of top 30% of frames	83.65%
Eliminate all “unknown” predictions	81.61%
Set minimum threshold on scores - 0.9	50.94%
Set minimum threshold on scores - 0.8	67.06%
Set minimum threshold on scores - 0.7	73.55%
Set minimum threshold on scores - 0.6	76.92%
Set minimum threshold on scores - 0.5	79.81%
Set minimum threshold on scores - 0.4	81.77%
Set minimum threshold on scores - 0.3	82.55%
Set minimum threshold on scores - 0.2	83.57%
Set minimum threshold on scores - 0.1	82.86%
1D-CNN (v.1)	85.05%
1D-CNN (v.2)	**87.01%**

Table 4–5: Comparison of late fusion methods. Accuracy results are based on their performance on the NHL-HPT dataset.

overall confidence score as frames with a lower confidence probability. This can affect the overall confidence score and hinder the accuracy of the overall label of the tracklet, as shown in our results in Table 5. This led us to experiment with other methods where we only considered a fixed number of frames in a tracklet with the highest confidence probabilities and ignored the rest with low confidence scores. By only taking the highest scores, we saw a slight improvement in accuracy with these methods over the average rule. Alternatively, we tried eliminating all confidence scores that were predicted as "unknown" and averaged the remaining ones to calculate the overall score, but this scheme saw a small decrease in accuracy. We also experimented with modifying the frames' confidence scores by applying a minimum threshold on every probability in the scores and setting all probabilities less than the threshold to zero, but this method offered little to no improvements to performance. There was a noticeable pattern where the performance became worse as the minimum threshold increased. This can be explained by the loss of more data that occurs as the minimum threshold increases, particularly when the confidences are not very high for any given tracklet example.

However, out of all the strategies we attempted, implementing a secondary 1D-CNN classifier to classify a tracklet's overall confidence score by treating it as a feature vector saw the greatest improvement in accuracy. Initially, we implemented a one-dimensional CNN (1D-CNN v.1) based on the LeNet [65] architecture and achieved a test accuracy of 85.05% on the NHL-HPT test set. This boost in performance to our approach encouraged us to experiment more with the 1D-CNN to further improve the performance. As mentioned earlier in the previous section, *Chapter*

3.3.1 Implementation Details, our 81-dimensional input vectors are relatively small in size. In most CNNs like LeNet, the volume of the data representation is usually downsampled spatially throughout the network. Since our particular data inputs are already spatially small, downsampling would cause a loss of data. We found that by maintaining the spatial size of the input vector representation throughout the 1D-CNN, we were able to improve its performance. To avoid the reduction of the vector representation's spatial size, we employed several means. First, we removed all intermediate max pooling layers since their main function in a network was to reduce the spatial size of the representation to reduce the number of parameters. In the convolutional layers, we used filter sizes of 1×3 with a stride of 1 and padded the vector representation by 1 on each side to maintain a spatial size of 1×81 until the end of the network. As well, we found that by replacing the final fully connected layer with 512 units with two fully connected layers each with 256 units, we were able to obtain better results. Due to the success we have encountered so far with using ResNets in this research, we also structured the convolutional layers in the 1D-CNN into two ResNet layer stacks, which helped us to achieve another small boost in performance. By implementing these changes into our initial 1D-CNN, we were able to increase the accuracy score of our approach to 87.01% with the final 1D-CNN (v.2). In Figure 4–4, we show the training and test accuracy curves during the training process. The test accuracy curve shows that it converged to 87.01% after 20,000 iterations.

For a closer analysis, we plot a confusion matrix (Figure 4–5) of the tracklet classification results after implementing a secondary 1D-CNN classifier to classify a

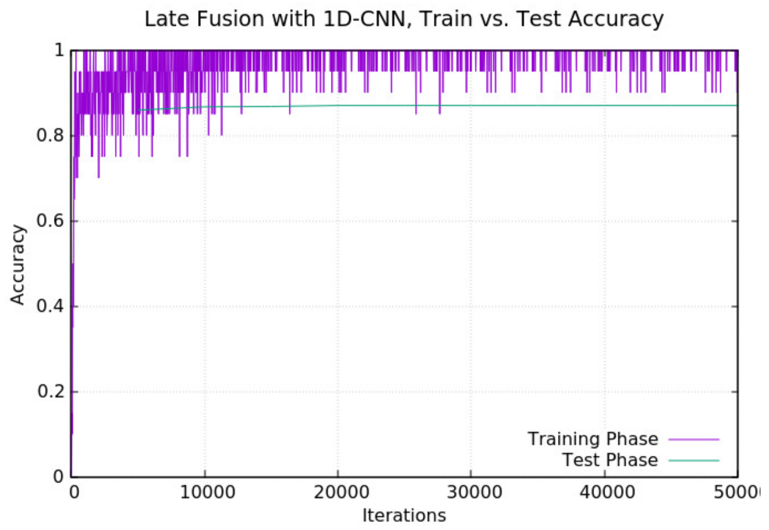


Figure 4–4: Training (purple) and test (green) accuracy plots of the final 1D-CNN (v.2)

tracklet's overall confidence score. A clear diagonal can be seen in the plot, which signifies correctly classified entries. A small percentage of false classifications can be seen scattered around the diagonal. Some jersey number classes experience more confusion than others, which can be due to a smaller number of training examples available for those classes. Noticeably, a few jersey numbers between 30 and 70 experience confusion with single digit numbers. During the classification process, one of the digits may have been more prevalent than the other throughout the tracklet. Overall, the confusion matrix shows a strong performance in hockey player identification, considering the limited size of our dataset.

4.4 Summary

In this chapter, we presented the experiment details and the results we obtained. We tested several different CNN architectures and investigated the advantages of

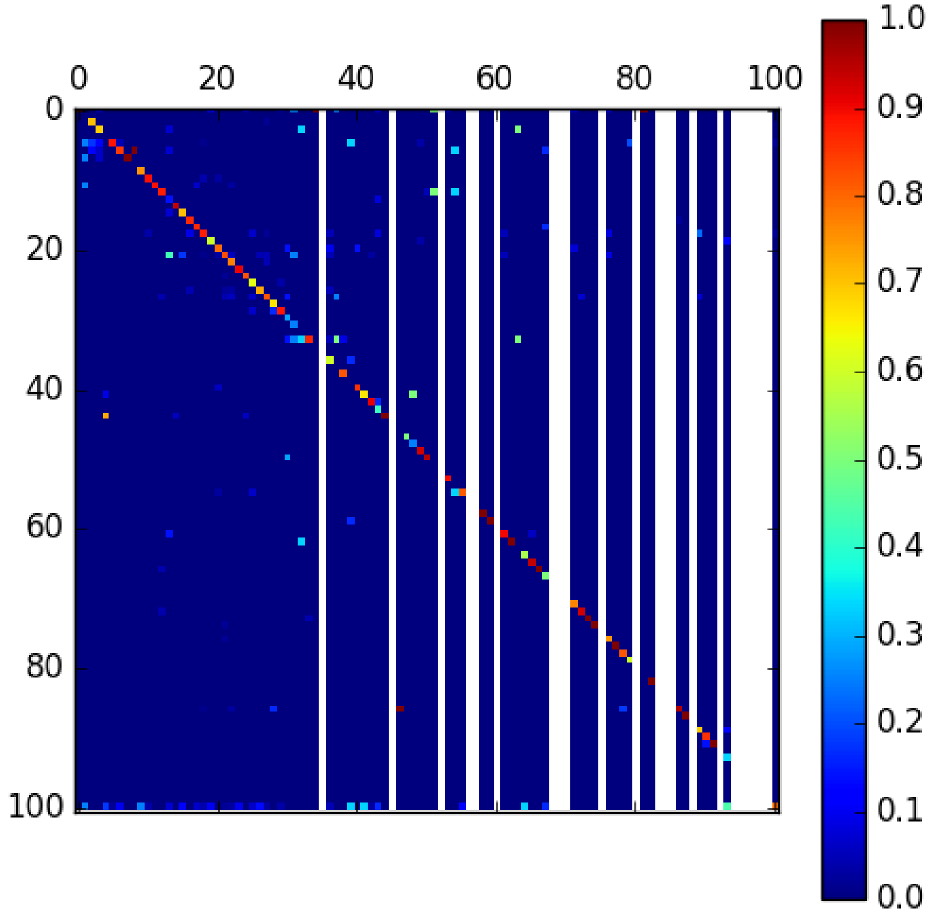


Figure 4–5: Confusion matrix of tracklet classification after implementing the ID-CNN late fusion classifier. The white bands in the matrix represent the jersey numbers that are not available in the NHL-HPT dataset.

implementing a recurrent module into CNN models and saw the benefits of some late score-level fusion methods. A summary of the player tracklet identification results of our models and the baselines are shown in Table 4–6 for comparison. With our proposed ResNet+LSTM model, we achieved an accuracy of 82.79% on the NHL-HPT test set after tuning the learning rate hyperparameter, which is an improvement over the baseline models. Compared to our ResNet+LSTM model, the ResNet-10 achieved 59.08% and the LRCN achieved 80.83% on the same test set. To further boost the performance of our approach, we employed an additional 1D-CNN classifier as a late score-level fusion technique. We used the 1D-CNN to classify the score output of the ResNet+LSTM network instead of relying on the maximum likelihood of the score probabilities to predict the player tracklet's label. By utilizing this secondary classifier, we were able to achieve an accuracy of 87.01%.

Models	**Test Accuracy**
ResNet-10 (baseline)	59.08%
LRCN (baseline)	80.83%
ResNet+LSTM	82.79%
ResNet+LSTM & 1D-CNN (v.2)	87.01%

Table 4–6: Overall methods comparison table. Accuracy results are based on their performance on the NHL-HPT test set.

CHAPTER 5
Conclusion

In this work, we address the task of hockey player identification in NHL broadcast videos and present our recurrent convolutional neural network based approach for jersey number recognition. Compared to previous work in sports player identification, we establish this task as a sequential problem involving multiple frames of a player as inputs. Our proposed approach is an end-to-end trainable network (ResNet+LSTM) that combines a ResNet model and LSTM layers to learn features of jersey numbers in any variable length player tracklet and allow the information from previous frames to flow across time-steps. This temporal pooling enables the model to make use of the spatio-temporal cues to produce an overall confidence score for a player tracklet. The recurrent nature of our model has shown to boost performance compared to the naive approach of operating on each frame individually. Additionally, we employ a late score-level fusion method of using a secondary 1-dimensional CNN classifier to evaluate the overall tracklet confidence score to achieve a higher classification accuracy score of 87.01%.

We also present the new NHL Hockey Player Tracklet (NHL-HPT) dataset with a total of 6,545 tracklet examples to train and evaluate the ResNet+LSTM network. This dataset features sequences of bounding boxes of hockey players and is the first of its kind, but unfortunately, it is not publicly available at this time.

For future work, we can integrate our method with the SPORTLOGiQ player detection and tracking algorithm for a fully automated player tracking and identification system for NHL broadcast hockey videos. This will make it possible to perform hockey player identification directly in the video and evaluate the robustness of our method when noisier player tracklets are used. To annotate future NHL games, we can incorporate prior knowledge, such as the team roster, to eliminate some of the false predictions.

References

[1] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, vol. abs/1512.03385, 2015.

[2] G. Huang, Z. Liu, and K. Q. Weinberger, "Densely connected convolutional networks," *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, vol. abs/1608.06993, 2016.

[3] J. Donahue, L. A. Hendricks, S. Guadarrama, M. Rohrbach, S. Venugopalan, K. Saenko, and T. Darrell, "Long-term recurrent convolutional networks for visual recognition and description," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. abs/1411.4389, 2014.

[4] W. Zaremba and I. Sutskever, "Learning to execute," in *arXiv:1410.4615*, vol. abs/1410.4615, 2014.

[5] I. J. Goodfellow, Y. Bulatov, J. Ibarz, S. Arnoud, and V. D. Shet, "Multi-digit number recognition from street view imagery using deep convolutional neural networks," in *arXiv:1312.6082*, vol. abs/1312.6082, 2013.

[6] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, pp. 1735–1780, Nov. 1997.

[7] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems 25* (F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, eds.), pp. 1097–1105, Curran Associates, Inc., 2012.

[8] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge," *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015.

[9] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," in *Proceedings of the 22Nd ACM International Conference on Multimedia*, MM '14, (New York, NY, USA), pp. 675–678, ACM, 2014.

[10] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," in *Computer Vision – ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part I* (D. Fleet, T. Pajdla, B. Schiele, and T. Tuytelaars, eds.), (Cham), pp. 818–833, Springer International Publishing, 2014.

[11] M. Simon, E. Rodner, and J. Denzler, "ImageNet pre-trained models with batch normalization," in *arXiv:1612.01452*, Dec. 2016.

[12] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Parallel distributed processing: Explorations in the microstructure of cognition, vol. 1," ch. Learning Internal Representations by Error Propagation, pp. 318–362, Cambridge, MA, USA: MIT Press, 1986.

[13] P. J. Werbos, "Generalization of backpropagation with application to a recurrent gas market model," *Neural Networks*, vol. 1, pp. 339–356, 1988.

[14] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *IEEE Transactions on Neural Networks*, vol. 5, pp. 157–166, Mar. 1994.

[15] R. Pascanu, T. Mikolov, and Y. Bengio, "Understanding the exploding gradient problem," in *arXiv:1211.5063*, 2012.

[16] M. Bertini, A. Del Bimbo, and W. Nunziati, "Player identification in soccer videos," in *Proceedings of the 7th ACM SIGMM International Workshop on Multimedia Information Retrieval*, MIR '05, (New York, NY, USA), pp. 25–32, ACM, 2005.

[17] M. Bertini, A. D. Bimbo, and W. Nunziati, "Matching faces with textual cues in soccer videos," in *2006 IEEE International Conference on Multimedia and Expo*, pp. 537–540, July 2006.

[18] S. Messelodi and C. M. Modena, "Scene text recognition and tracking to identify athletes in sport videos," *Multimedia Tools Appl.*, vol. 63, pp. 521–545, Mar. 2013.

[19] M. Šaric, H. Dujmic, V. Papic, N. Rožic, and J. Radic, "Player number recognition in soccer video using internal contours and temporal redundancy," in *Proceedings of the 10th WSEAS International Conference on Automation & Information*, ICAI'09, (Stevens Point, Wisconsin, USA), pp. 175–180, World Scientific and Engineering Academy and Society (WSEAS), 2009.

[20] Q. Ye, Q. Huang, S. Jiang, Y. Liu, and W. Gao, "Jersey number detection in sports video for athlete identification," *SPIE Conference Proceedings*, vol. 5960, pp. 1599–1606, 07 2005.

[21] P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*, vol. 1, pp. I–511–I–518 vol.1, 2001.

[22] J. Matas, O. Chum, M. Urban, and T. Pajdla, "Robust wide-baseline stereo from maximally stable extremal regions," *Image and Vision Computing*, vol. 22, no. 10, pp. 761 – 767, 2004. British Machine Vision Computing 2002.

[23] A. Sato and K. Yamada, "Generalized learning vector quantization," in *NIPS*, 1995.

[24] A. Khotanzad and Y. H. Hong, "Invariant image recognition by zernike moments," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 12, pp. 489–497, May 1990.

[25] S. Messelodi and C. Modena, "Automatic identification and skew estimation of text lines in real scene images," *Pattern Recognition*, vol. 32, no. 5, pp. 791 – 810, 1999.

[26] C.-W. Lu, C.-Y. Lin, C.-Y. Hsu, M.-F. Weng, L.-W. Kang, and H.-Y. M. Liao, "Identification and tracking of players in sport videos," in *Proceedings of the Fifth International Conference on Internet Multimedia Computing and Service*, ICIMCS '13, (New York, NY, USA), pp. 113–116, ACM, 2013.

[27] W.-L. Lu, J.-A. Ting, K. P. Murphy, and J. J. Little, "Identifying players in broadcast sports videos using conditional random fields," *CVPR 2011*, pp. 3249–3256, 2011.

[28] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *Int. J. Comput. Vision*, vol. 60, pp. 91–110, Nov. 2004.

[29] W. L. Lu, J. A. Ting, J. J. Little, and K. P. Murphy, "Learning to track and identify players from broadcast sports videos," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, pp. 1704–1716, July 2013.

[30] S. Gerke, K. Müller, and R. Schäfer, "Soccer jersey number recognition using convolutional neural networks," in *2015 IEEE International Conference on Computer Vision Workshop (ICCVW)*, pp. 734–741, Dec 2015.

[31] S. Gerke, A. Linnemann, and K. Müller, "Soccer player recognition using spatial constellation features and jersey number recognition," *Computer Vision and Image Understanding*, vol. 159, pp. 105 – 115, 2017. Computer Vision in Sports.

[32] J. D. Lafferty, A. McCallum, and F. C. N. Pereira, "Conditional random fields: Probabilistic models for segmenting and labeling sequence data," in *Proceedings of the Eighteenth International Conference on Machine Learning*, ICML '01, (San Francisco, CA, USA), pp. 282–289, Morgan Kaufmann Publishers Inc., 2001.

[33] K. P. Murphy, Y. Weiss, and M. I. Jordan, "Loopy belief propagation for approximate inference: An empirical study," in *Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence*, UAI'99, (San Francisco, CA, USA), pp. 467–475, Morgan Kaufmann Publishers Inc., 1999.

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

[35] J. Liu, J. Luo, and M. Shah, "Recognizing realistic actions from videos x201c;in the wild x201d;," in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1996–2003, June 2009.

[36] J. C. Niebles, C.-W. Chen, and L. Fei-Fei, "Modeling temporal structure of decomposable motion segments for activity classification," in *Proceedings of the 11th European Conference on Computer Vision: Part II*, ECCV'10, (Berlin, Heidelberg), pp. 392–405, Springer-Verlag, 2010.

[37] J. Sivic and A. Zisserman, "Video google: A text retrieval approach to object matching in videos," in *Proceedings of the Ninth IEEE International Conference on Computer Vision - Volume 2*, ICCV '03, (Washington, DC, USA), pp. 1470–, IEEE Computer Society, 2003.

[38] H. Wang, M. M. Ullah, A. Klaser, I. Laptev, and C. Schmid, "Evaluation of local spatio-temporal features for action recognition," in *BMVC 2009 - British Machine Vision Conference* (A. Cavallaro, S. Prince, and D. Alexander, eds.), (London, United Kingdom), pp. 124.1–124.11, BMVA Press, Sept. 2009.

[39] P. Dollar, V. Rabaud, G. Cottrell, and S. Belongie, "Behavior recognition via sparse spatio-temporal features," in *Proceedings of the 14th International Conference on Computer Communications and Networks*, ICCCN '05, (Washington, DC, USA), pp. 65–72, IEEE Computer Society, 2005.

[40] I. Laptev, M. Marszalek, C. Schmid, and B. Rozenfeld, "Learning realistic human actions from movies," in *2008 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1–8, June 2008.

[41] K. Soomro, A. Roshan Zamir, and M. Shah, "UCF101: A Dataset of 101 Human Actions Classes From Videos in The Wild," in *arXiv:1212.0402*, Dec. 2012.

[42] S. Abu-El-Haija, N. Kothari, J. Lee, A. P. Natsev, G. Toderici, B. Varadarajan, and S. Vijayanarasimhan, "Youtube-8m: A large-scale video classification benchmark," in *arXiv:1609.08675*, 2016.

[43] S. Ji, W. Xu, M. Yang, and K. Yu, "3d convolutional neural networks for human action recognition," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 35, pp. 221–231, Jan. 2013.

[44] M. Baccouche, F. Mamalet, C. Wolf, C. Garcia, and A. Baskurt, "Sequential deep learning for human action recognition," in *Proceedings of the Second International Conference on Human Behavior Unterstanding*, HBU'11, (Berlin, Heidelberg), pp. 29–39, Springer-Verlag, 2011.

[45] Y. Qin, L. Mo, J. Ye, and Z. Du, "Multi-channel features fitted 3d cnns and lstms for human activity recognition," in *2016 10th International Conference on Sensing Technology (ICST)*, pp. 1–5, Nov 2016.

[46] D. Tran, L. Bourdev, R. Fergus, L. Torresani, and M. Paluri, "Learning spatiotemporal features with 3d convolutional networks," in *Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV)*, ICCV '15, (Washington, DC, USA), pp. 4489–4497, IEEE Computer Society, 2015.

[47] A. Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar, and L. Fei-Fei, "Large-scale video classification with convolutional neural networks," in *Proceedings of the 2014 IEEE Conference on Computer Vision and Pattern Recognition*, CVPR '14, (Washington, DC, USA), pp. 1725–1732, IEEE Computer Society, 2014.

[48] K. Simonyan and A. Zisserman, "Two-stream convolutional networks for action recognition in videos," in *arXiv:1406.2199*, vol. abs/1406.2199, 2014.

[49] M. Baccouche, F. Mamalet, C. Wolf, C. Garcia, and A. Baskurt, "Action classification in soccer videos with long short-term memory recurrent neural networks," in *Proceedings of the 20th International Conference on Artificial Neural Networks: Part II*, ICANN'10, (Berlin, Heidelberg), pp. 154–159, Springer-Verlag, 2010.

[50] J. Y. Ng, M. J. Hausknecht, S. Vijayanarasimhan, O. Vinyals, R. Monga, and G. Toderici, "Beyond short snippets: Deep networks for video classification," in *arXiv:1503.08909*, vol. abs/1503.08909, 2015.

[51] Z. Wu, X. Wang, Y.-G. Jiang, H. Ye, and X. Xue, "Modeling spatial-temporal clues in a hybrid deep learning framework for video classification," in *ACM Multimedia*, 2015.

[52] Z. Wu, Y.-G. Jiang, X. Wang, H. Ye, and X. Xue, "Multi-stream multi-class fusion of deep networks for video classification," in *Proceedings of the 2016 ACM on Multimedia Conference*, MM '16, (New York, NY, USA), pp. 791–800, ACM, 2016.

[53] N. McLaughlin, J. M. d. Rincon, and P. Miller, "Recurrent convolutional network for video-based person re-identification," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1325–1334, June 2016.

[54] L. Wu, C. Shen, and A. van den Hengel, "Deep recurrent convolutional networks for video-based person re-identification: An end-to-end approach," in *arXiv:1606.01609*, vol. abs/1606.01609, 2016.

[55] Y. Yan, B. Ni, Z. Song, C. Ma, Y. Yan, and X. Yang, "Person re-identification via recurrent feature aggregation," vol. abs/1701.06351, 2017.

[56] R. Hadsell, S. Chopra, and Y. LeCun, "Dimensionality reduction by learning an invariant mapping," in *Proceedings of the 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition - Volume 2*, CVPR '06, (Washington, DC, USA), pp. 1735–1742, IEEE Computer Society, 2006.

[57] K. Cho, B. van Merrienboer, Ç. Gülçehre, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using RNN encoder-decoder for statistical machine translation," in *arXiv:1406.1078*, vol. abs/1406.1078, 2014.

[58] T. Ojala, M. Pietikäinen, and T. Mäenpää, "Multiresolution gray-scale and rotation invariant texture classification with local binary patterns," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 24, pp. 971–987, July 2002.

[59] O. Chapelle and S. S. Keerthi, "Efficient algorithms for ranking with svms," *Inf. Retr.*, vol. 13, pp. 201–215, June 2010.

[60] H. W. Kuhn, "The hungarian method for the assignment problem," *Naval Research Logistics Quarterly*, vol. 2, no. 1-2, pp. 83–97, 1955.

[61] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "SSD: Single Shot MultiBox Detector," in *arXiv:1512.02325*, Dec. 2015.

[62] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A Large-Scale Hierarchical Image Database," in *CVPR 2009*, 2009.

[63] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng, "Reading digits in natural images with unsupervised feature learning," in *NIPS Workshop on Deep Learning and Unsupervised Feature Learning 2011*, 2011.

[64] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson, "How transferable are features in deep neural networks?," in *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'14, (Cambridge, MA, USA), pp. 3320–3328, MIT Press, 2014.

[65] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," in *Proceedings of the IEEE*, pp. 2278–2324, 1998.