Where Is the Puck?

Tiny and Fast-Moving Object Detection in Videos

Xionghao Yang, Department of Electrical and Computer Engineering

McGill University, Montreal, Quebec, Canada

December, 2020

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

Table of Contents

Table of Contents ..... *ii*

Acknowledgements ..... *iv*

Abstract ..... *v*

Abrégé ..... *vii*

Chapter 1 Introduction *1*

Chapter 2 Related Work *5*

2.1 Sports Analytics **5**

2.2 Object Detection **8**

2.2.1 Object Detection in Still Images 9

2.2.2 Object Detection in Videos 11

Chapter 3 Methodology *13*

3.1 Dataset Creation **13**

3.1.1 Data Collection and Description 13

3.1.2 Dataset Analysis and Statistics 16

3.2 Model Structure **18**

3.2.1 Stage One: Extracting Representations of Video Frames 19

3.2.2 Stage One: Extracting Representations of Video Frames 21

3.2.3 Stage Two: Computing the Temporal Features 31

3.3 Evaluation Metrics **35**

Chapter 4 Experiments *40*

4.1 Implementation Details **40**

4.2 Training and Inference **41**

4.2.1 Training 42

4.2.2 Inference 48

4.3 Backbones **50**

4.4 Multi-task Loss **52**

4.5 Resolution **57**

4.6 Summary **60**

Chapter 5 Results and Discussion *62*

References*..*75

Acknowledgements

Firstly, I would like to thank my supervisor, Prof. Martin Levine of the Center for Intelligent Machines at McGill University, for his guidance, patience, and insistence on high standards through each stage of this research project. Prof. Levine supported and encouraged me when I was experiencing hard times. He is more than a supervisor to me. Words cannot express how grateful I am for being his student.

Secondly, I would like to thank my colleagues in the Intelligent Multimodal Video Analysis research lab for the helpful discussions and happy memories. Finally, I would like to thank my families and friends for the unconditional love and support.

Abstract

Ice hockey, also known as hockey in North America, is considered as the official national winter sport in Canada. With the development of computer vision technologies, the emergence of powerful computing devices, and the advancement of personal entertainment equipment, sports analytics have become one of the most popular and potential research areas. The key to automatically understand and analyze a hockey game is to find the location of the "protagonist", the puck. Therefore, in this project, we aim to solve the problem of tracking the puck in NHL broadcasting videos using a tiny object detector. Since the previous works mainly focused on detecting regular-size objects in still images, our project is a novel contribution in the field of object detection that opens doors to potential research and application. One of the main challenges of this project is the tiny size of the puck. The puck in the broadcast videos is approximately 4.3^2 pixels wide, which accounts for less than 0.005% of the area of a frame. However, the average size of the small objects in MS COCO [1] is 28^2 pixels, which accounts for around 0.255% of the area of images. This comparison shows the difficulty of our underlying task. Additionally, the motion blur caused by the fast movement of the puck, the occlusion between the puck and other objects, and the visual noises (e.g., the advertisements on the rink) introduce more challenges to the project. To tackle these challenges, we first collect high-resolution clips from different NHL games and manually annotate the puck in these clips. This dataset, McGill Puck Dataset (MPD), is the first public ice hockey dataset with precise puck annotation. We then design a two-stage tiny object detection model. Stage One is a two-dimension convolutional neural network that extracts summarized representation of each frame. In Stage Two, we stack and feed the fusion of these per frame representations to a three-dimension convolutional neural network to decode the temporal information from the video. The objective of this detector is only to find the center point of the puck (i.e., without extracting the full shape of the puck), which converts the traditional object detection problem to a keypoint detection task. With this modified objective, the evaluation metrics used in other

object detection tasks (e.g., computing Intersection over Union) are not applicable anymore. Instead, we measure the L2 distance between the predicted and ground-truth center points to determine the prediction's correctness. Overall, our tiny and fast-moving object detector achieves the Precision of 0.908 and the Recall of 0.867. The F1 score reaches 0.887, which is significantly higher than the performance of YOLOv3 [2] (F1 score = 0.685) and Mask RCNN [3] (F1 score = 0.749) on the McGill Puck Dataset.

Abrégé

Le hockey sur glace, ou tout simplement appelé « le hockey » en Amérique du nord, est considéré comme le sport national d'hiver au Canada. Le développement de la vision numérique, l'émergence des appareils informatiques de pointe et l'avancement des équipements de divertissements personnels ont rendu l'analyse sportive un domaine de recherche particulièrement populaire. La clé pour comprendre et analyser un match de hockey de manière automatisée est de localiser le « protagoniste » du jeu, plus précisément, la rondelle. Pour cela, dans ce projet, nous cherchons à résoudre le problème du suivi de la rondelle dans les vidéos diffusées par la LNH en utilisant un détecteur d'objets miniatures. Les travaux précédents dans le champ de la détection des objets étaient surtout focalisés sur la détection d'objets de taille régulière dans des images fixes. Ainsi, notre projet est une nouvelle contribution qui ouvre des portes à des recherches et applications potentielles dans le domaine. Un des défis majeurs dans cette étude est la petite taille de la rondelle. En effet, dans les vidéos diffusées, la rondelle a une largeur de 4.3^2 pixels, ce qui représente moins de 0.005% de l'aire de l'image. En revanche, la taille moyenne des petits objets dans MS COCO [1] est de 28^2 pixels, ce qui correspond à environ 0.255% de l'aire de l'image. Cette comparaison permet de souligner la complexité de notre tâche. De plus, certains facteurs comme le flou de mouvement dû au déplacement rapide de la rondelle, l'occlusion entre cette dernière et d'autres objets, ainsi que le bruit visuel (à savoir les publicités dans la patinoire) introduisent des défis supplémentaires à considérer. Pour adresser ces problématiques, premièrement, nous avons recueilli des vidéos de haute résolution de différents matchs de la LNH et avons annoté manuellement la rondelle dans ces clips. Ce jeu de donnée, le *McGill Puck Dataset* (MPD), est la première base de données publique sur le hockey comprenant une annotation précise de la rondelle. Ensuite, nous avons conçu un modèle de détection d'objets minuscules en deux étapes. La première étape comprend un réseau de neurones convolutif bidimensionnel qui extrait une représentation résumée de chaque image. Dans la deuxième étape, nous empilons et alimentons la fusion de ces représentations par image dans un réseau neuronal convolutif tridimensionnel pour décoder la dimension temporelle de la vidéo. Le but de ce détecteur est de trouver uniquement le point central de la rondelle (sans extraire sa

forme complète), ce qui transforme le problème de détection d'objets traditionnels en une tâche de détection de points-clés. Tenant compte de cette modification, les métriques d'évaluation utilisés dans d'autres tâches de détection d'objets (par exemple : l'évaluation de l'Intersection sur Union) ne s'appliquent plus. Par conséquence, nous mesurons plutôt la distance L2 entre les points centraux prédits et les points de vérité-terrain pour déterminer l'exactitude de la prédiction. Globalement, notre détecteur d'objets minuscules et rapides atteint une Précision de 0.908 et un Rappel de 0.867. Le F1 atteint une valeur de 0.887, ce qui est nettement supérieur à la performance de YOLOv3 [2] (valeur F1 = 0.685) et de Mask RCNN [3] (valeur F1 = 0.749) dans le*McGill Puck Dataset.*

Chapter 1 Introduction

Ice hockey, also known as the national winter sport in Canada, is considered as the game of "fire and ice". The sport is known to be fast-paced and physical, with teams fielding six players at a time: one goaltender and five players who skate the span of the ice trying to control the puck and score goals against the opposing team [4]. No other sports can even come close to hockey in terms of speed, which leads to plenty of fierce physical confrontations in a game. With the progress of sports commercialization, the popularity of ice hockey has greatly increased in recent years. This has led to a strong demand for efficient hockey analytics. The availability of sports analytics enables hockey fans to get a more in-depth understanding of the game. It can also help to lower the entry threshold for "newbies" who desire to enjoy an ice hockey game. By benefiting from precise sports analytics, professional players and their teams can make specific training plans to enhance competition. Additionally, the game commentators can give the audience a more comprehensive picture with the support of these sports analytics data.

To understand an ice hockey game thoroughly, we need to monitor the trajectory of players and the puck on the rink, the actions of players, and the relationship between players and the puck. Among all the variables, locating the puck is the most essential because the objective of most players is to control the puck. Thus, the location of the puck is the key to understanding the events happening on the ice. The objective of our project is to design an algorithm which can automatically find the puck in an ice hockey game.

Finding the puck in a video is a typical object detection task. Object detection is one of the most fundamental and well-developed fields in computer vision. In terms of the input data format (image and video), the algorithms can be grouped into two subsets: still image object detection and video object detection. In the former, there are state-of-art models like Faster RCNN [5], FCN [6], and YOLO [7]. The detection accuracy and speed have steadily improved over

the last decade. However, the data used to train and test these models only consist of normal-sized objects (e.g. people, dog, car). Therefore, these detection models cannot address tiny object detection. With the increased availability of computational capability, more exploration of video object detection has been performed. Naïve video object detection is conducted by splitting a video into frames and detecting the objects in each image (frame). By adequately utilizing temporal information, the model can refine the detection results from keyframes to other frames, which can significantly improve the detection speed. Instead of aiming at accelerating the detection, we want to improve the detection accuracy of tiny objects by leveraging the temporal information from videos. That is why we need to design a new model focusing on finding extremely small objects in videos.

The significant challenges of puck detection include the tiny size, motion blur, occlusion, and visual noise. In standard cases, there is no occlusion or background distraction, and the appearance of the puck is not hazy. However, the area of the puck in a 720 x 1280 broadcast video is approximately 45 pixels, which accounts for less than 0.005% of the area of a frame. As shown in Table 1.1, the objects in PASCAL VOC [8], cover on average 20.85% of the area of the whole input image. To compare models of different size objects, the MS COCO dataset [1] collects objects with different sizes. The average-size object in MS COCO accounts for only 4.32% of the total area of the input image. The relative size of objects in the commonly-used dataset is considerably larger than the relative size of the puck. The extremely tiny size leads to less significant features and more potential disturbance, which brings the degree of difficulty of the underlying task to another level.

Dataset	McGill Puck Dataset	Pascal VOC [8]	MS COCO [1]
Average object size	0.005%	20.85%	4.32%

Table 1.1 Average object size in different datasets

Additionally, when players pass the puck, it moves at a significantly high speed, which creates severe motion blur. The shape of the puck is likely to be hazy because fast-moving objects appear as semi-transparent streaks larger than their original size. Furthermore, players

and sticks might occlude the puck due to different camera angles. Temporal information embedded in the video sequence can be helpful for the occlusion case. Moreover, the visual noise of the advertisements painted on the side-boards and the ice could be a severe distraction.

In this thesis, we discuss the design of a multi-frame video object detection model, which can automatically detect tiny objects in short video clips. In the first stage, a 2D convolutional neural network is used to extract the representation of each frame. In the second stage, we stack nine consecutive representations and feed them into a 3D convolutional neural network to compute the temporal information. Finally, three one-layer convolutional blocks are implemented to predict the center points' heatmap, the shape of the object, and the center points' offsets. The whole model follows the "backbone + multiple heads" structure [9]. After exploring different convolutional neural network backbones in the first stage, we chose the DLA (Deep Layer Aggregation) network.

To select the best parameters and model architectures, we need evaluation metrics to compare model performance. The so-called confusion matrix is a table that is widely used to describe the performance of a supervised learning model on a set of test data. In order to create the table, we need to determine the correctness of the predictions. As for object detection, the most common method is to measure the overlap of the predicted and ground-truth bounding boxes. This evaluation metric is known as Intersection over Union (IoU), which is the value of dividing the area of the overlap by the area of the union. However, this evaluation metric cannot be implemented in this project. Due to the tiny size of the object and the possible motion blur, the IoU value is not able to precisely reflect the performance. Therefore, we employ another evaluation metric for tiny object detection, which measures the L2 distance between the center points of the prediction and ground truth. By setting a distance threshold, the prediction is considered correct if the distance is less than the threshold.

Three significant contributions are presented in this thesis to solve the tiny and fast-moving object detection problem.

1. We explored a new research direction in the field of object detection, which focuses on detecting tiny and fast-moving objects in videos. Although parts of the groundwork [10] have already been done, the new research branch is still a big challenge. Looking further ahead, tiny object detection can benefit the development of many potential applications, such as sports analytics, surveillance, missile tracking, etc.
2. The objective of this project is to automatically locate the puck in broadcast hockey games. In order to achieve this goal, we collected a custom dataset, the McGill Puck Dataset (MPD), in which the puck was manually annotated. This dataset is used for model training, model selection, and model evaluation. We also compared the performance of other state-of-art object detectors on MPD.
3. After a thorough exploration of different model structures, we designed a two-stage detector which can precisely find objects in videos despite such challenges like as tiny size, motion blur, occlusion, and visual noise. Our tiny and fast-moving object detector achieves the Precision of 0.908 and the Recall of 0.867. The F1 score reaches 0.887, which is significantly higher than the performance of YOLOv3 [2] (F1 score = 0.685) and Mask RCNN [3] (F1 score = 0.749) on the McGill Puck Dataset.

The remainder of this thesis is organized as follows. Chapter 2 briefly introduces state-of-art object detection methods and the latest application of sports analytics, especially the one related to ice hockey. Then we thoroughly explain our model and training strategy in Chapter 3. The experiments and their results are shown in Chapter 4. Finally, in Chapter 5, we discuss the strengths and weaknesses of our model and list the areas of research in which we are interested.

Chapter 2 Related Work

In recent decades, sports analytics has drawn researchers' attention and has been widely used in major sports like soccer, tennis, basketball, etc. A large portion of professional clubs has benefited from the services of professional statisticians to develop their tactics and strategies. Coaches can use these statistics to specify and optimize a training program for individual players.

The credit for the increasing interest in sports analytics belongs to the developments in deep learning, especially in computer vision. Many productive computer vision algorithms focus on different tasks, such as object detection, object tracking, and action recognition. These algorithms can potentially be applied to sports analytics. For example, in a soccer game, all players and the ball's locations can be precisely predicted with the support of an object detector. The ice hockey analytics are currently underdeveloped due to the challenge of detecting the objects in complicated game scenes. The puck is the key to understand an ice hockey game. However, due to the challenges like tiny size, motion blur, occlusion, there is no mature algorithm which can accurately detect the puck in videos. In order to fill in this knowledge gap, we aim to design a video object detection model that can function in complex situations, for example, puck detection.

2.1 Sports Analytics

The practice of using sports analytics has been around for decades, but recent advances in data collection and management technology have broadened its scope. The use of data and statistics has become prolific throughout most major sports. For example, by automatically detecting the location of players and tennis balls, tennis matches can be analyzed, and the skills of the players can be quantitatively evaluated [11]. Similarly, a tracking system can be applied to find the location of a soccer ball, which can assist fans to understand the game better [12].

A large portion of professional teams now routinely draw on the services of professional statisticians to support their operations. Analytics has many on-field applications in a sports environment, including managing both individual and group performance. These teams can use data to optimize exercise programs for their players and develop nutrition plans to maximize fitness.

For now, and in the future, using machine learning in sports analytics is essential. Firstly, the primary advantage of using machine learning is its flexibility in managing large volumes of data as well as multiple data sources. The use of machine learning has had the effect of making data collection more accessible and more effective. Secondly, machine learning promotes a quicker method of obtaining analytics. By using analytical results, coaches and sports analysts may not need to spend as many hours watching recorded game films because meaningful insights can be derived by merely viewing the machine learning analysis. Thirdly, with the development of machine learning, the technology has exceeded human performance in many aspects. This trend gives us confidence that machine learning can become the dominant tool in sports analytics.

Machine learning analytics is quickly gaining popularity in the world of sports. Several sports have recorded massive success through the use of machine learning. For example, the famous Manchester City Football Club has employed the use of machine learning analysis [13]. Accordingly, this has dramatically improved their team performance. The Manchester City Football Club is also known to apply this technique in recruiting new players and determining what play will suit a specific field position. Machine learning analysis is also used by the Chicago-based firm STATS (Sports Team Analysis and Tracking Systems) [14] in assembling information on various player movements. In previous years, STATS has employed the use of cameras in European soccer stadiums and NBA arenas. These cameras, which are integral to the SportsVU system [15], are programmed to track the movement of individual players and the ball at 25 frames per second. STATS cameras have also been programmed to analyze different players' strengths and weaknesses, such as running speed, ball possession and distance/endurance.

The development of sports analytics in ice hockey is slow due to the lack of technical support to tackle complicated scenes. The researchers are more interested in some more basic and low-level tasks, such as player tracking, puck detection, and action recognition. Among all of these tasks, player tracking and action recognition have been developed most successfully. The state-of-art object detection algorithms [16, 17, 18] and person Re-Identification tools [19, 20] enable us to keep tracking the players on the ice. Likewise, the action recognition algorithms [21, 22] can capture the player's actions on the ice. In contrast, while puck detection has been the earliest research direction, it remains the one with the slowest progress. In 1996, the FoxTrax System [28] debuted as a unique tool to track every movement of the puck and superimpose a bluish glow around the puck for television viewers. To use the FoxTrax system, a small circuit board, which contains infrared emitters and a shock sensor, is installed in the puck. In a hockey game, once active, the puck would emit infrared pulses, which in turn were received by a complicated detection system in the arena. The detection system of FoxTrax contains more than twenty detectors and ten modified cameras which have to be placed in the rafters in the arena in which the game is being held in order to receive the pulse. The drawbacks of this puck tracking system are apparent. First, a large amount of specific-designed and high-standard pucks needs to be manufactured. Second, due to the limitation of battery capacity, the puck needs to be changed regularly. Third, the pulse receiving systems need to be installed in all arenas. The maintenance cost can increase the burden to all professional ice hockey teams and broadcasting platforms. The FoxTrax system was the first successful tryout of automatic puck detection, but was eventually made obsolete due to the above reasons. The exploration of puck detection has never stopped since then. However, there was no impressive breakthrough until the increased interest in deep learning object detection algorithms had successfully lead to various applications. In 2019, PuckNet [10], proposed by the team of John Zelek, can automatically estimate the puck location in a broadcast video using deep learning computer vision technologies and without any assistance of a physical sensor. The broadcast video was "cut" into 1-second clips and input into a 3D CNN which produced an output heatmap of the puck.

The heatmap indicated the probabilities of the occurrence of the puck at each pixel. The PuckNet[10] also mentions that a task-specific model evaluation method is necessary for puck detection, which differs from traditional object detection evaluation methods. However, the data used to train PuckNet only consisted of approximate puck locations. Instead of creating a custom dataset that contained the exact puck locations for every frame, it leverages the existing play-by-play hockey event annotations to obtain the puck annotations. The puck tends to move at a significant speed, which means the puck location will change remarkably in a short period. This is why using the event location to represent the puck location is inadequate. In contrast, in this thesis, we seek to design a method that continuously and accurately detects and tracks the puck in raw broadcast videos, thereby enhancing the following process of event detection.

2.2 Object Detection

The objective of object detection is to solve the most fundamental problem in computer vision: ***what objects are where?*** A successful detection model can automatically predict the location and category of the desired object. As one of the earliest considered problems in computer vision, object detection forms the basis of many other applications, such as object tracking, instance segmentation, image captioning, etc. There are different subsets under object detection that are based on model structure and data format. Images and videos are the two most frequently used data formats in this research. In the beginning, because of the lack of powerful computational support, most detection algorithms can only be applied on still images. The development of object detection in the still image domain is partitioned into two stages according to the usage of deep learning techniques, which leads to remarkable breakthroughs. There are two mainstream machine learning object detection models in terms of model architecture (more details in Section 2.2.1): "one-stage detector" and "two-stage detector". Encouraged by applications like sports analytics and surveillance, more research on video object detection has been conducted. Video is not only a sequence of images; it is instead a sequence

of related images. This rich information embedded in the video brings tremendous challenges to analyzing video data with acceptable speed. Nevertheless, temporal information embedded in the videos can boost the accuracy or accelerate the detection. The three kinds of state-of-the-art video object detection methods based on different approaches are post-processing, multi-frame, and optical field methods (more details in Section 2.2.2).

2.2.1 Object Detection in Still Images

Due to the "rebirth" of the convolutional neural network in 2012, the performance of object detection has improved dramatically. The so-called deep convolutional network can now automatically compute high-level representations of objects, which used to be manually extracted in traditional object detectors. In Figure 2.2.1, there are two types of state-of-art object detectors: “one-stage detector” and “two-stage detector” [24].

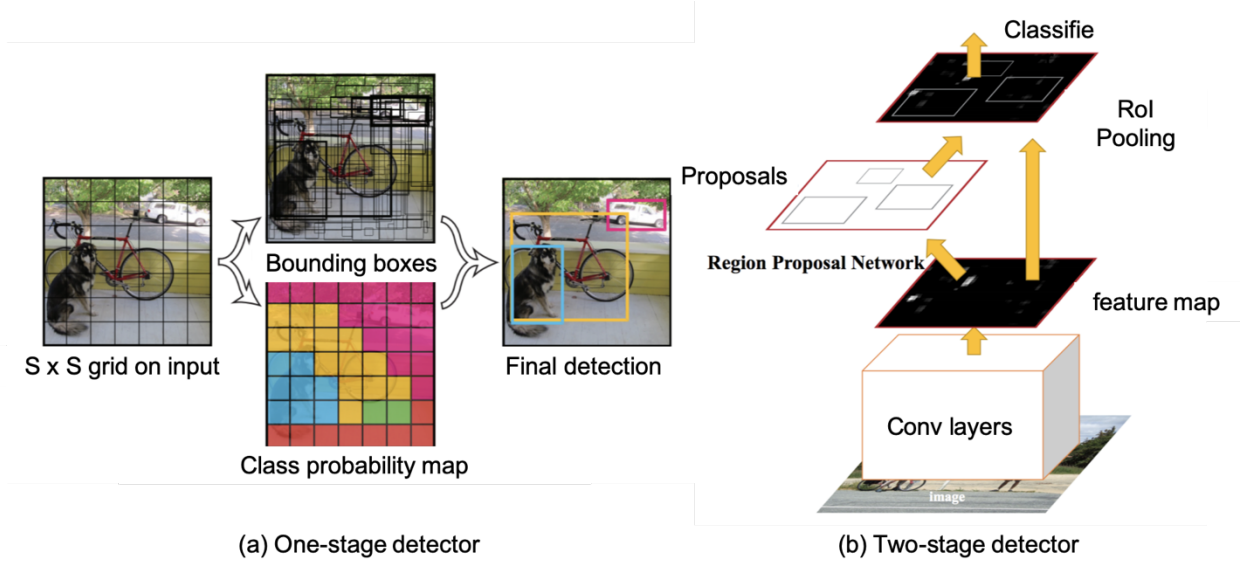

Figure 1.2.1 [5,7] Comparison of the one-stage and two-stage detector. The left graph shows a typical one-stage object detection model (You Only Look Once [7]) which treats object detection as a simple regression problem by taking an input image and learning the class probabilities and bounding box coordinates. The right graph represents the a typical two-stage object detection model (Faster RCNN [5]). Firstly, it extracts region proposals. Then, it classifies the regions of interest (RoIs), also known as proposals, accordingly.

Two-stage detectors use a region proposal block to generate regions of interest in the first stage. The proposals are sent to individual networks for object classification and bounding-box regression. The Region-based Convolutional Neural Networks (RCNN) [25], introduced by Ross Girshick, starts with the extraction of a set of RoIs using a selective search algorithm [26]. Each RoI is rescaled into a same fixed-sized image and input into a convolutional neural network to extract a high-dimensional representation. Finally, the representations of each RoI are fed into class-specific linear support vector machines to predict the object's presence and classify the object categories.

Compared with the traditional detector, RCNN significantly improves accuracy by involving convolutional neural networks. However, the drawbacks of RCNN are apparent. It extracts nearly two thousand RoIs for each image and inputs them into a convolutional neural network one by one, which brings redundant computations on a large number of overlapped proposals. SPPNet [27] solves this drawback by adding a Spatial Pyramid Pooling (SPP) layer that generates a fixed-length feature representation regardless of the size of images. This upgrade enables SPPNet to compute the representation of the whole input image, and directly pulls out the feature representations of RoIs from the intact feature map. Fast RCNN [28] was proposed in 2015 as a combination of RCNN and SPPNet. In Fast RCNN, an object classifier and a bounding box regressor are simultaneously trained. Nevertheless, the detection speed and accuracy of Fast RCNN still cannot meet the requirements from practical applications. In order to get more fast and more accurate detection, Region Proposal Network (RPN) proposes to replace the selective search algorithm in Faster RCNN [5] to extract the RoIs. RPN, as an independent block, optimizes the entire detection processing, which remarkably improves the detection accuracy and speed. Object detectors only leverage the information from the top layer before the occurrence of Feature Pyramid Network (FPN) [29]. The deep layers contain more semantic information, while the shallow layers contain more intuitional information of shape, appearance, and location. FPN is a pioneer work that indicates the importance of feature fusion. Inspired by FPN, more researcher start to fuse the features from different layers to compute

more summarized features which can significantly improve the detection accuracy. Though FPN outperforms other object detectors in the aspects of accuracy and speed, it still can't be considered a real-time detector. Such two-stage models reach the highest accuracy rates but are typically slow. In order to achieve real-time detection, researchers abandoned the two-stage detecting mode.

Instead of considering the whole task as a classification plus regression problem, the 'one-stage' models upgrade object detection as a simple regression problem by taking an input image and end-to-end learning the class probabilities and bounding box coordinates. This innovation leads to the improvement of testing speed in small cost of accuracy. YOLO (You Only Look Once) [7] is the first and the most successful one-stage detector in the deep learning detection era. Because it simplifies the whole detection problem as a regression problem, it can provide inference at an incredible speed. As a one-stage object detector, YOLO is fast. Still, it is not good at recognizing irregularly shaped objects or small objects due to a limited number of bounding box candidates.

2.2.2 Object Detection in Videos

The most pioneering work in computer vision has focused on image processing, while the video processing field has been less deeply explored. A video is essentially a sequence of images (frames). However, this definition cannot encapsulate the whole idea of what video processing is, and that is because video processing adds a new dimension to the problem: the temporal dimension. The expensive computation of training and inference is the major reason why the growth of video object detection is much slower than still image object detection. With the upgrade of computational hardware and the release of many well-annotated video object datasets, more and more research concentrating on videos has been launched. There are three different kinds of mainstream video object detection methods: post-processing and multi-frame methods.

In post-processing methods, an extra block is added to concatenate the predictions in

separate frames. These algorithms (e.g., Seq-NMS [30]) apply a modification of the post-processing phase that uses high-scoring object detections from nearby frames to boost scores of weaker detections within the same clip. Given a video sequence of region proposals and their corresponding class scores, Seq-NMS associates bounding boxes in adjacent frames using a simple overlap criterion. It then selects boxes to maximize a sequence score. The chosen boxes are then used to suppress overlapping boxes in their respective frames and are subsequently rescored to boost weaker detections.

Instead of treating the video as separate frames, the multi-frame detectors consider the video clip as an entirety. They input the whole video clip into the model. An end-to-end 3D convolutional neural network is designed for object detection and segmentation in videos [31]. In the 3D CNN based video object detection model, a video is first divided into equal-length clips. Then, the 3D features of these video clips are extracted with the model. Finally, the temporal object detection is performed using proposals generated from the 3D features. Due to the high volume of computation, the processing time cannot be as fast as real-time (30 fps or higher) at the current state [31]. The stability, as well as the precision of the detections, can be improved by the 3D convolution as the architecture can effectively leverage the temporal dimension altogether (aggregating features between frames). Furthermore, using a recurrent neural network, such as Long Short Term Memory (LSTM), can also achieve multi-frame video object detection.

In conclusion, if we address the puck detection as a still image object detection task, we cannot leverage temporal information. Due to the property of the puck (tiny size and fast movement), current still image object detection models are not able to find representative features of the tiny and fast-moving object. Thus, in order to find the puck in those challenging scenes, temporal information is essential. In contrast, current video object detection algorithms focus on utilizing temporal information to accelerate the detection procedure for videos instead of refining frame-based detections with temporal information. Hence, we aim to design an algorithm that can leverage the videos' temporal information to achieve frame-based detection.

Chapter 3 Methodology

3.1 Dataset Creation

As discussed above, tiny and fast-moving object detection is a novel research branch in the field of computer vision. There is no satisfactory dataset available for this specific task, so we collected and annotated a custom dataset: McGill Puck Dataset (MPD). In this dataset, the broadcast videos[1] are manually cut into short clips that do not contain sharp camera angle changes. We record the location and status of the puck in each frame precisely. MPD is the first open-source video dataset containing objects with such a small size and high moving speed.

3.1.1 Data Collection and Description

Supervised learning models can better provide robustness by involving training data from different lighting situations and with visual noise. MPD contains 130 video clips extracted from 5 NHL games. After preprocessing and cutting the whole match into pieces, we manually annotated the puck in video clips, frame by frame. The information of each sample is recorded using two lists. As shown in Table 3.1.1.1, the first list includes necessary information about the image, such as resolution, original game source, file name, and image ID. The second list covers the information about the objects in the image, which is considered as ground truth used to train, validate, and test models. The accurate position of each object is manually obtained using a rectangle that precisely covers the whole object area. The rectangle, also known as the bounding box, is marked with its top-left corner coordinates, width, and height. As shown in Figure 3.1.1.1, to annotate the puck, we firstly zoom into the region where the object is and then draw a bounding box covering the objects precisely. The bounding box's information is automatically stored in a JSON file. The tool used for annotation is Computer Vision Annotation

[1] All the NHL game videos are downloaded from *https://nhl66.ir* and should not be used for any commercial purposes.

Tool (CVAT) [33], a powerful interactive video and image annotation tool for computer vision, which allows users to save annotation results in a custom format.

		Normal Example	Occlusion Example	Blur Example
Images	Width	1280	1280	1280
	Height	720	720	720
	Source	Game_001	Game_002	Game_004
	File Name	Val_0024_00065	Val_0001_00055	Val_0020_00111
	Image ID	2793	65	2251
Annotations	Is_visible	1	1	1
	Is_blurred	0	0	1
	Is_occluded	0	1	0
	Area	21.81	40.05	100.95
	Category ID	1	1	1
	Annotation ID	2210	55	1767
	Bounding Box	[717.21, 308.76, 5.99, 3.64]	[928.26, 410.74, 6.99, 5.73]	[214.88, 467.90, 14.22, 7.10]

Table 3.1.1.1 Examples of annotations of different cases

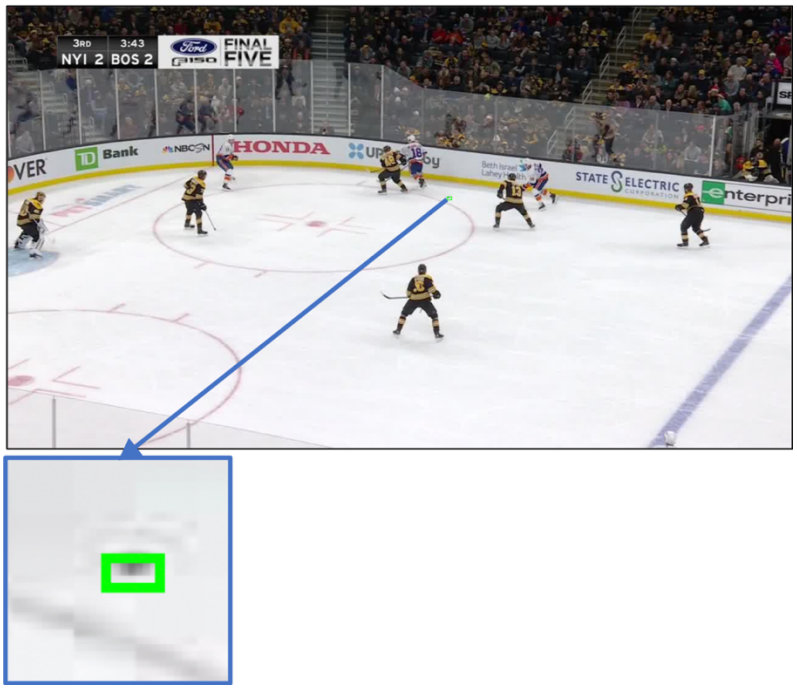

Figure 3.1.1.1 Illustration of how to annotate the puck in a game frame. The green bounding box shows the ground-truth location of the puck in this frame. In order to closely observe the results, we zoom in on the regions that contain ground-truth puck (shown in blue windows

under the original frame).

Besides the location of the object, its status is also recorded based on the following exploration. We classify an object into one of four groups according to its visibility and relationship with other objects as follows: normal object, occluded object, blurry object, and invisible object. Three binary digits (is_visible, is_occluded, and is_blurred) are used to indicate an object's status. There is no overlap between the four classes, which means that an object can have at most one status. We provide strict criteria to determine an object's status. The object will be seen as occluded when other objects partially block it in the image, or it disappears within less than three frames (0.05 seconds):

$$occluded\ object \rightarrow "partially\ occluded" \vee "disappearing\ time < 0.05\ seconds"$$

Occluded objects don't have a significant feature representations as objects under normal circumstances. Thus, we should leverage the information from neighboring frames to infer the location of the occluded puck. Motion blur is the streak-like effect that occurs when shooting a video, because the objects are moving rapidly through the frame, or the camera exposure is particularly long. The appearance of the blurring objects could be shifted, which increases the difficulty to the detection. Motion blur is inevitable in sports broadcast, so that involving more blurring samples in the dataset can effectively improve the model's performance. The object is marked as blurry when its shape distortion happens due to rapid movements:

$$blurry\ object \rightarrow shape\ distortion$$

Figure 3.1.1.2 and Table 3.1.1.1 show two occluded and blurred examples with corresponding annotations. Due to the camera rotation, the object can "disappear" from the scene for a while (more than 0.05 seconds). In this case, the location of the objects cannot be correctly deduced, so we neglect the bounding box and record the status of the object as invisible. All the objects not belonging to one of these three situations are considered as normal cases.

$$invisible\ object \rightarrow disappearing\ time \geq 0.05\ seconds$$

$$normal\ object \rightarrow (\neg occluded\ object) \wedge (\neg blurry\ object) \wedge (\neg invisible\ object)$$

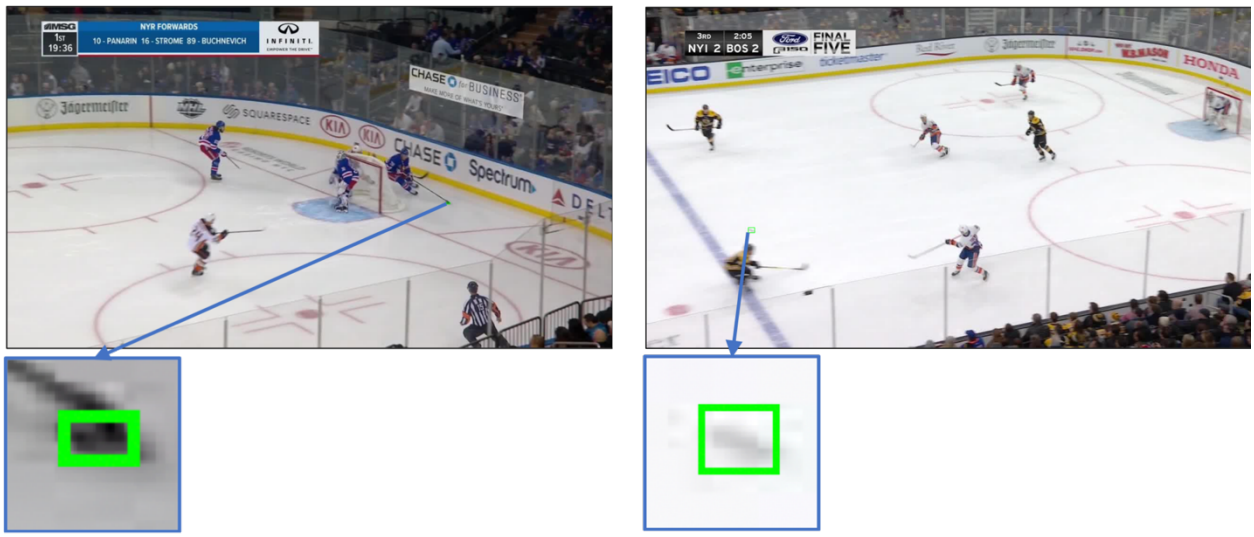

Figure 3.1.1.2 Illustration of how to annotate an occluded (a) and blurry object (b). The green bounding boxes show the ground-truth and predicted locations of the puck in this frame. In order to closely observe the results, we zoom in on the regions that contain the ground-truth pucks(shown in blue windows under the original frame).

3.1.2 Dataset Analysis and Statistics

In MPD, all the raw game videos have a resolution of 1280 x 720 and FPS of 60 (sixty frames per second). Typically, in a video object detection task, lower resolution videos are preferred. Higher-resolution input means more computational complexity. However, since the tiny size of the puck, higher-resolution video data are essential because they provide more detailed information, that can help to distinguish the puck. In MPD, there are 130 clips and 17997 frames in total. These data are partitioned into the training set, validation set, and testing set in the ratio of 8:1:1. As Figure 3.1.2.1 illustrates, 54.7% of the frames contain objects with normal status. Frames that include occluded and blurry objects account for 17.3% and 4.0%, respectively. The remaining 24.0% are “blank data”, meaning that there are no visible objects in these frames. Thus, more investigation and discussion about how the data’s imbalance affects the performance needs to be conducted.

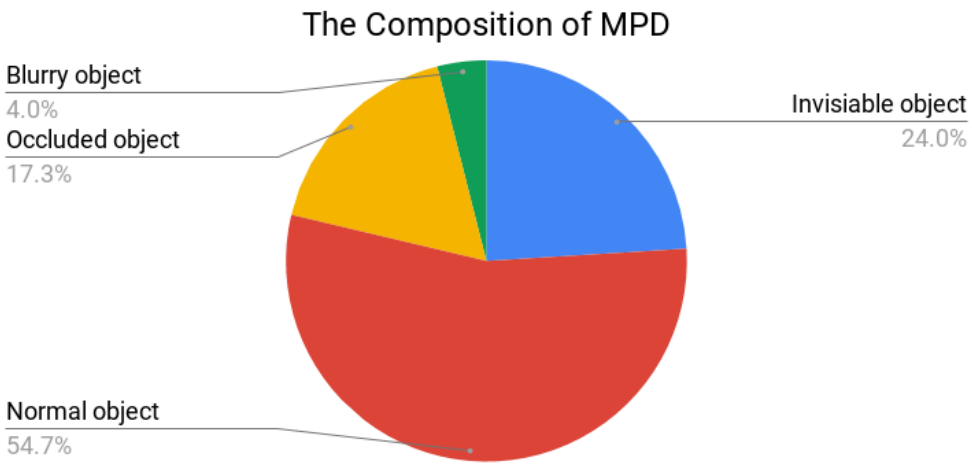

Figure 3.1.2.1 The proportion of the four different kinds of status in MPD

The average size of objects in MPD is 45 pixels, which accounts for less than 0.005% of the full image size. To highlight how tiny the objects in MPD are, we compare with other popular object detection datasets in terms of the objects' absolute and relative size. Absolute size is merely the number of pixels the object covers, while the relative size is the absolute size divided by the total number of pixels in the video frames:

$$absolute\ size = \#\ of\ covered\ pixels$$

$$relative\ size = \frac{\#\ of\ covered\ pixels}{\#\ of\ total\ pixels} \times 100\% = \frac{absolute\ size}{frame's\ resolution} \times 100\%$$

In PASCAL VOC, an object's average absolute size is 37,343 pixels, while the average image resolution is 179,024 pixels. MS COCO collects various sized objects intentionally. An object's average size is 11,917 while the average resolution of the covering area is 275,832 pixels. The relative size of objects in PASCAL VOC and MS COCO is 20.85% and 4.32%, respectively, which are 4000 and 800 times larger than data in MPD, respectively. Figure 3.1.2.2 depicts the distribution of the data in these three datasets in terms of their relative size of objects. Most of the PASCAL VOC objects account for 10% - 50% of the input image. Almost no sample contains an object smaller than 0.1%. In MS COCO, the relative size of more than one-third of objects is between 0.1% and 1%. There are still approximately 20% objects in MS COCO that are smaller than 0.1%, but practically no object's size is less than 0.01%. In contrast, 94.29%

of data in MPD belong to 'less than the 0.01% category. The comparison explicitly shows that the objects in MPD have extremely tiny sizes. This property makes MPD fill up the research blank left by other mainstream datasets in the field of object detection. More research focusing on tiny object detection can be conducted with the help of this dataset.

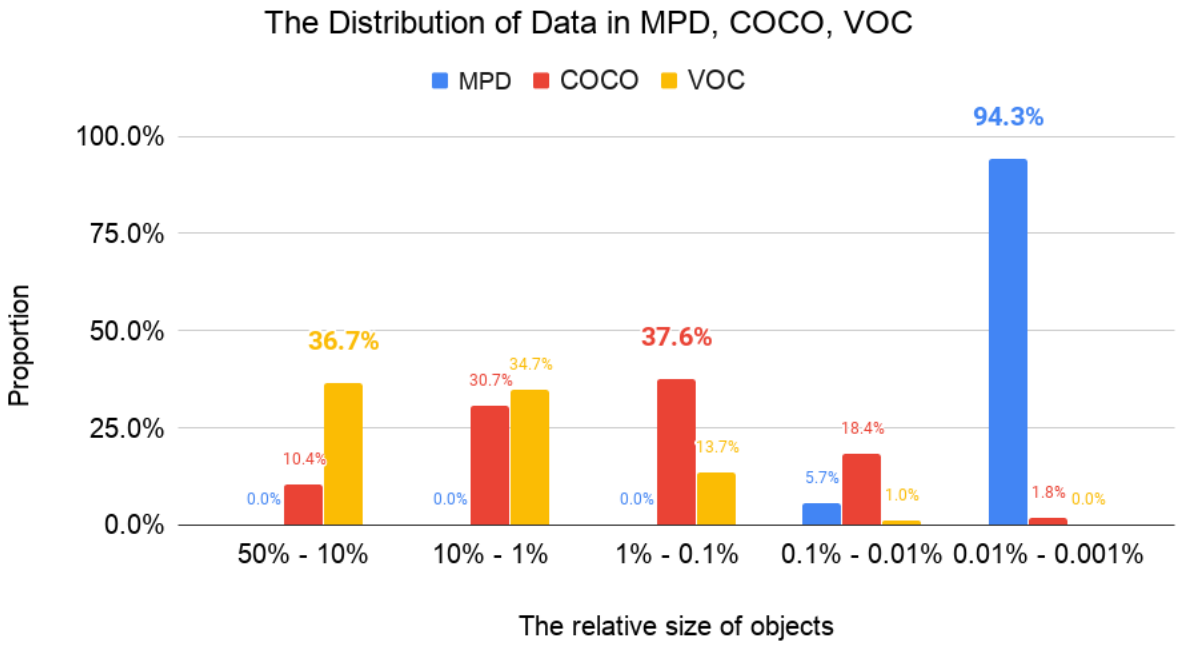

Figure 3.1.2.2 The distribution of data in different datasets in terms of their relative areas

3.2 Model Structure

In this section, we introduce the structure of the model designed to detect tiny and fast-moving objects in videos. Before discussing the detailed information about every component, the overall structure of the model is briefly demonstrated in Section 3.2.1. After that, we go deeper into all stages and present their mathematical descriptions (Section 3.2.2 and 3.2.3). In addition, the ideas and training schemes used to improve the final performance are also explained in the following content.

3.2.1 Stage One: Extracting Representations of Video Frames

In order to detect the puck in videos, we need to distinguish it from the background and other objects. The most reasonable way to do this is to extract the features, also known as the representations, only owned by the target object. Object detection is then tackled by finding a particular region in the input, which has similar features to the target object. For example, when it comes to the puck, we generally assume that the puck's color is black, the shape is round, and the size is small. Thus, we recognize the regions with similar features as predicted objects. Each manually-designed feature corresponds with an intuitive property of the object. The puck's appearance in individual frames can change significantly due to disturbances like visual noise from background, motion blur, camera zoom-in and zoom-out. Besides, most manually-designed features are low-dimensional, which leads to the detectors being either too general or too over-engineered. The puck's appearance in individual frames can change significantly due to disturbances like visual noise from the background, motion blur, camera zoom-in and zoom-out. Thus, using human-designed features to find the puck in broadcast videos is not efficient nor efficacious.

Since the development of convolutional neural networks, researchers have tended to use CNNs to extract the features of objects automatically. CNNs work like black boxes because researchers cannot tell what the specific meanings of the output of each convolutional layer are. However, using CNNs as feature engines enables the model to extract representative features, including both local features (appearance) and global features (relationships with other objects). Applications [2, 3, 5, 6, 7] have proved the outstanding performance of CNNs as feature engines in the field of computer vision. Therefore, in this project, we apply modified CNNs to extract each video clip's representations and predict the location of the puck based on these representations. The videos (D x C x W x H) have one more dimension than still image data (C x W x H). Because of the complexity and large size of video data, the detection model is resolved into two stages. In other words, we first focus on extracting the representation of each frame. Then, we stack the representations of consecutive frames and compute the temporal features.

The final representation of each frame contains both the local and temporal features. Finally, we apply three parallel stages (called blocks in CNN parlance) to predict the object's existence and location using the final representation of each frame.

Following the above ideas, we design a two-stage model (shown in Figure 3.2.1.1) that simultaneously leverages both semantic and temporal information to detect the puck in broadcast videos. The objective is to locate the puck in a particular frame (t = T). In Stage One, we use a 2D CNN to extract each frame's representation. To extract the temporal features of the frame (t = T), we have to make sure there are at least four sequential frames before and after it, respectively. Only qualified frames are input into the next stage. In Stage Two, we stack feature maps computed from nine continuous frames (from t = T - 4 to t = T + 4) as the new input data. The structure of Stage Two is inspired by the famous "backbone + multi-heads" structure [9]. The backbone network is responsible for extracting a representation that contains higher level summarized features, while each head uses this representation as input to predict its desired outcome. The backbone of Stage Two is a 3D CNN, which upgrades the frame's representation by adding temporal features. Then, three individual convolutional blocks further process the output of the 3D CNN to predict the heatmap of center points, the shape of objects, and the offsets of center points. The center points' heatmap depicts the possibility of each pixel being an object's center point. The shape of objects is predicted at each pixel corresponding to each center point. In order to compensate for the deviation caused by downsampling and upsampling operations, we introduce the offsets to refine the center points' locations. The ultimate prediction is generated by selecting the center points with high possibilities, retrieving their corresponding appearance information, and slightly adjusting the location according to their offsets.

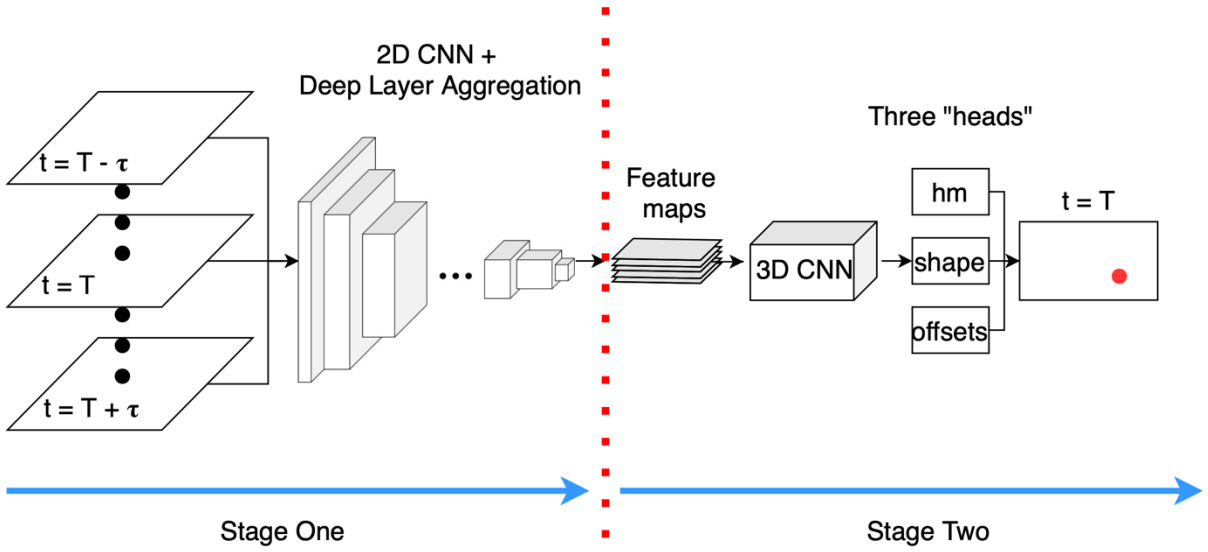

Figure 3.2.1.1 The two-stage model used to detect tiny and fast-moving objects in videos

In the following context, we use diagrams and mathematical expressions to illustrate each stage's detailed structure. Also, the ideas used to improve the regular convolutional networks and schemes used to train each part of the model are further explained.

3.2.2 Stage One: Extracting Representations of Video Frames

In Stage One, video clips are firstly split into frames and input into a deep 2D convolutional neural network to extract the representation of each frame. Then, all the representations are sorted and stacked together. Finally, these intermediate products are input into Stage Two to enable the extraction of the temporal features.

The objective of Stage One is to summarize a high-level representation of each frame individually. Therefore, we apply a modified 2D CNN as the feature engine in this stage. The structure of the convolutional network is inspired by Deep Layer Aggregation (DLA) [34]. By virtue of this unique feature fusion technique, the extracted representation preserves better local and global information from both shallow and deep layers.

3.2.2.1 Feature Fusion

State-of-art object detection models have implemented deeper and wider convolutional neural

networks. Networks from LeNet [35] through VGG16 [36] to ResNet [37] stack layers and blocks in sequences. Meanwhile, the depth and width of the networks are further expanded, which brings challenges for optimization and computation. Layerwise accuracy comparisons [38] show that deeper layers extract more semantic and global features, but the experiments do not prove that the last layer is the ultimate representation for any task. Skip connections between adjacent layers have been proven effective for more structured tasks [38, 39]. However, the performance reached a bottleneck because these models only use the top-layer information to do the ensuing classification and bounding box regression. With the emergence of FPN [29], feature fusion has provided more potential solutions to complex object detection applications. Visual recognition requires rich representations that span levels from low to high, scales from small to large, and resolution from fine to coarse. Compounding and aggregating the feature representations from different levels improves the inference of what and where. Thus, exploring the best way to aggregate the layers and blocks in a regular convolutional network deserves further attention.

Aggregation in a convolutional neural network is defined as the combination of the output of the individual layers. Since networks contain many layers and connections, modular design helps counter complexity by grouping and repetition. Layers are grouped into blocks, which are then grouped into stages according to the feature resolution. The aggregation operations are typically conducted between blocks and stages. Without any feature fusion, the data would flow sequentially, and the output would need to be computed from the last block only (shown in Figure 3.2.2.1 (a)). The deeper stages provide more semantic but spatially coarser features. In shallow aggregation models (shown in Figure 3.2.2.1 (b)), the appearance information from superficial layers plays a more critical role than semantic information from deep layers. The simple skip connections from shallow to deep stages merge scales and resolution, but they aggregate the shallowest layers the least.

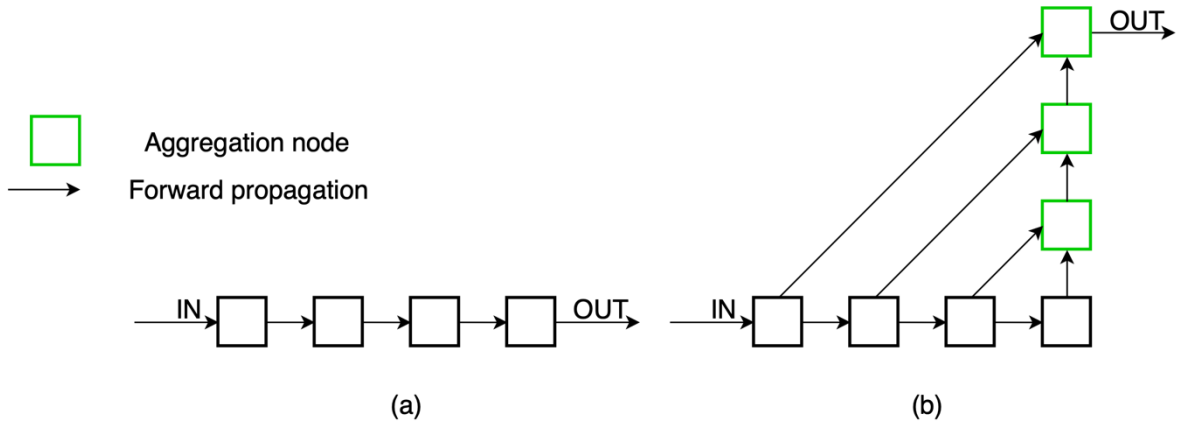

Figure 3.2.2.1 No aggregation and shallow aggregation structures [34]

Iterative deep aggregation (IDA) [34] follows the sequential stacking of backbone architecture. Instead of shallowly connect the layers, IDA aggregates the features from different depths progressively. Aggregation starts at the shallowest layers and then iteratively merges features from deeper layers. IDA enables shallow features, which contain more information about the object's appearance, to be propagated through the whole network. Iterative aggregation refines the output features of each layer equally, while shallow aggregation "brutally" combines features from all layers. The drawback of iterative aggregation is that it still keeps the sequential structure, making it insufficient for fusing a great number of blocks in a network. The IDA structure (shown in Figure 3.2.2.2) can be mathematically described as:

$$\mathrm{I}(x_1, x_2, \dots, x_n) == \begin{cases} x_1 & if\ n = 1 \\ I(N(x_1, x_2), \dots, x_n) & otherwise, \end{cases} \quad (1)$$

where N is the aggregation node.

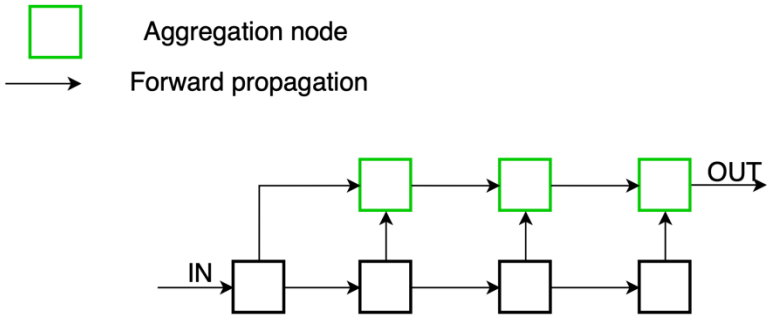

Figure 3.2.2.2 Iterative deep aggregation structure [34]

Hierarchical deep aggregation (HDA) [34] abandons the sequential network structure and aggregate all blocks in a tree structure. In HDA, the shallow and deep layers are combined in many more various ways to span more of the feature hierarchy. One clever design feature in HDA is that it feeds the output of an aggregation node back into the backbone as the input to the next sub-tree, which propagates the aggregation of all previous blocks instead of preceding block alone to better preserve features. The HDA structure (shown in Figure 3.2.2.3) can be mathematically denoted as:

$$T_n(x) = N(R^n_{n-1}, R^n_{n-2}, \dots, R^n_1, L^n_1(x), L^n_2(x)), \quad (2)$$

where N is the aggregation node. R and L are defined as:

$$L^n_2(x) = B(L^n_1(x)), L^n_1(x) = B(R^n_1(x))$$

$$R^n_m(x) = \begin{cases} T_m(x) & if\ m - n = 1 \\ T_m\left(R^n_{m+1}(x)\right) & otherwise, \end{cases}$$

where B represents a convolutional block.

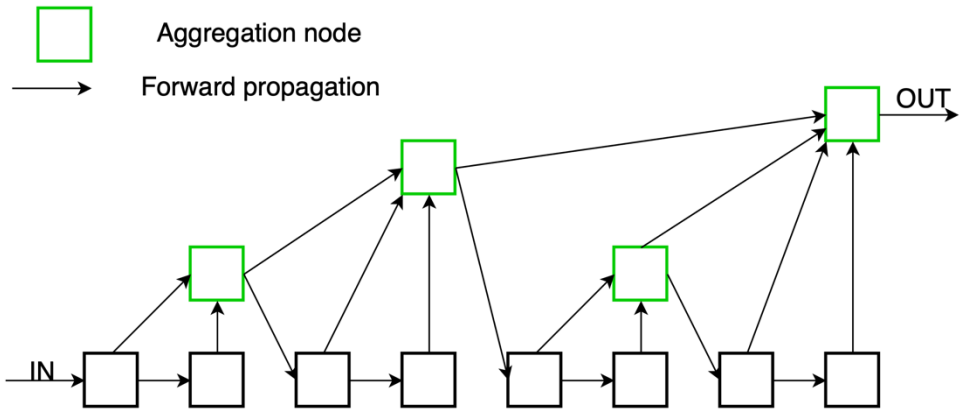

Figure 3.2.2.3 Hierarchical Deep Aggregation structure [34]

Theoretically, an aggregation node can contain arbitrary operations, but for simplicity, we use a single 3 × 3 convolution followed by a batch normalization and an activation function. The aggregation nodes in IDA are always binary, while HDA nodes can have a variable number of arguments depending on the depth of the tree structure. According to previous research [34], the residual connection between nodes can help HDA when the tree has four levels or more. The basic aggregation node can be denoted as:

$$N(x_1, x_2, \dots, x_n) = \sigma(BN(\sum_i W_i x_i + b))$$

where σ is the activation function, and BN is the batch normalization process. Adding the residual connection can prevent problems like vanishing gradients and curse of dimensionality. The revised aggregation node can be denoted as:

$$N(x_1, x_2, \dots, x_n) = \sigma\left(BN\left(\sum_i W_i x_i + b\right) + x_n\right),$$

DLA can be considered a combination of IDA and HDA, enabling the deep network to extract the global and local representations from the network's different levels. In DLA, the stages are connected with IDA, and the layers are connected with HDA within each stage. This type of aggregation is easily achieved by sharing aggregation nodes. Finally, to complete, we

merge Equation (1) and (2) in the root node of each hierarchy. Between stages, max-pooling layers with kernel size two and stride two are implemented. The end of every stage halves the resolution, giving six stages in total, with the first stage maintaining the input resolution while the last stage is 32 times downsampled. A basic six-stage DLA is shown in Figure 3.2.2.4.

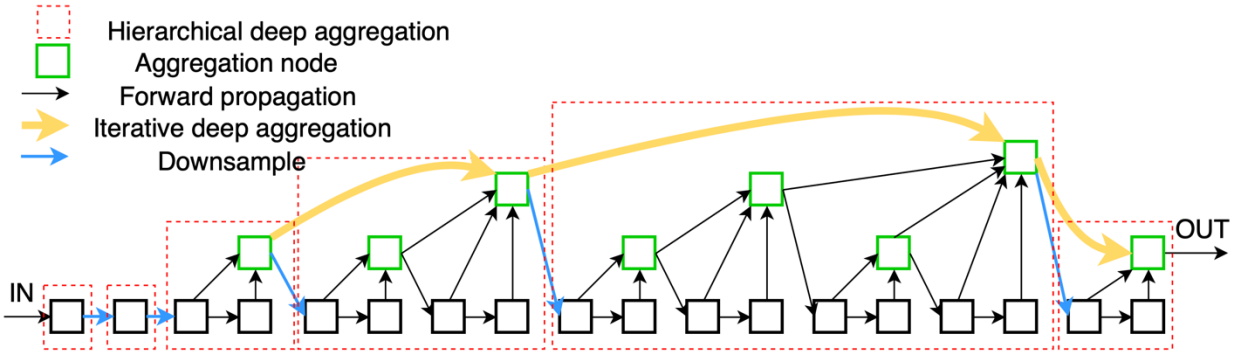

Figure 3.2.2.4 Basic Deep Layer Aggregation structure [34]

The resolution of the feature map drops sharply in the deeper stages. For example, in a basic 6-level DLA network, the feature map's size is scaled down to 1/32 of the original image size at the last stage. Nevertheless, the combination of IDA and HDA enables the model to preserve the location information from shallow layers to a better extent. When we retrieve the prediction results for the original resolution, the output from the last stage has a greater effect because of its low resolution. In order to get the desired output resolution, we add a series of iterative aggregation nodes and upsampling nodes to fuse the features from stages in the basic DLA. The extra iterative aggregation increases both depth and the resolution by projection and upsampling, where all parameters can be jointly learnt during the optimization of the network. As shown in Figure 3.2.2.5, a 6-stage basic DLA is used first to compute a representation with different scales. The resolutions of the output of all six stages are 512 x 512, 256 x 256, 128 x 128, 64 x 64, 32 x 32, 16 x 16. After storing the outputs from the basic DLA, a series of iterative aggregations is applied from the back to the front until we obtain the representation with the desired resolution. The upsampling scale in the process is always two. Eventually, we once again iteratively aggregate it with previous intermediate products from shallow to deep, which further

refine the features. In the upgraded DLA, we use IDA three times: to connect stages in the basic DLA, to retrieve the representation with the desired resolution, and to further improve feature quality.

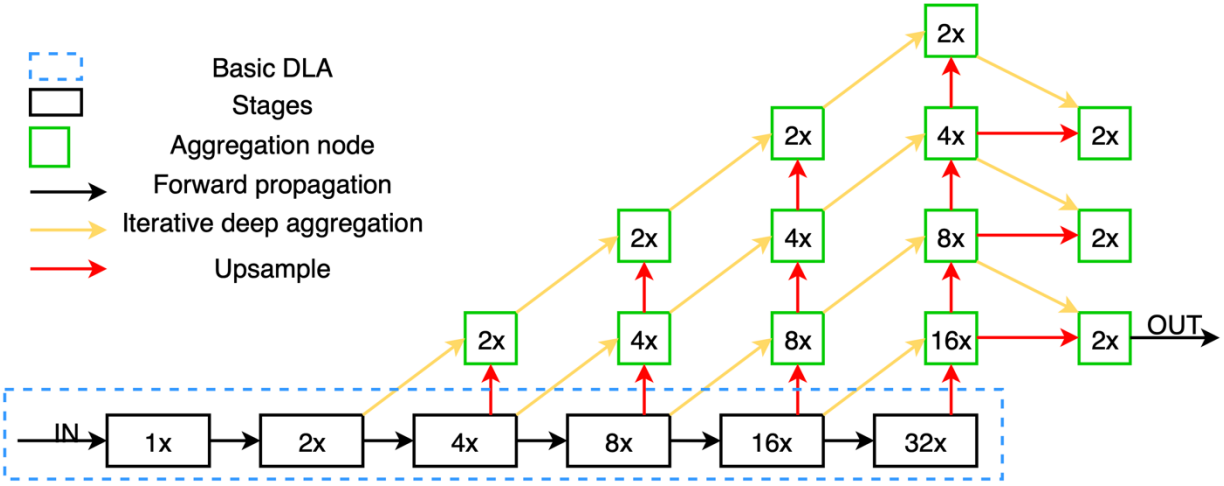

Figure 3.2.2.5 The upgraded Deep Layer Aggregation structure

3.2.2.2 Structure

Figure 3.2.2.6 illustrates the structure of Stage One and the add-on used to train this stage individually. The input data of Stage One is a set of three-channel frames. All frames are rescaled to 512 x 512 by default. The rescaled RGB images are input into a 16-channel convolutional layer (kernel size = 7) followed by a batch normalization layer [40] and the ReLU activation function [41]. The core of this stage is the DLA network, which is used to extract the representation of each frame. The basic DLA is used to compute the intermediate representations with different resolutions. It contains six levels, which means that the smallest representation's resolution is 512 x 16 x 16. According to the downsampling ratio in configuration (R = 4), we add iterative aggregation nodes to fuse and upsample all intermediate representations to extract the features with the ultimate desired resolution (64 x 128 x 128). We then propagate this feature forward to aggregate with two smaller intermediate features (128 x 64 x 64, 256 x 32 x 32) from the previous step. The refined output of the DLA network is

a 64 x 128 x 128 feature map.

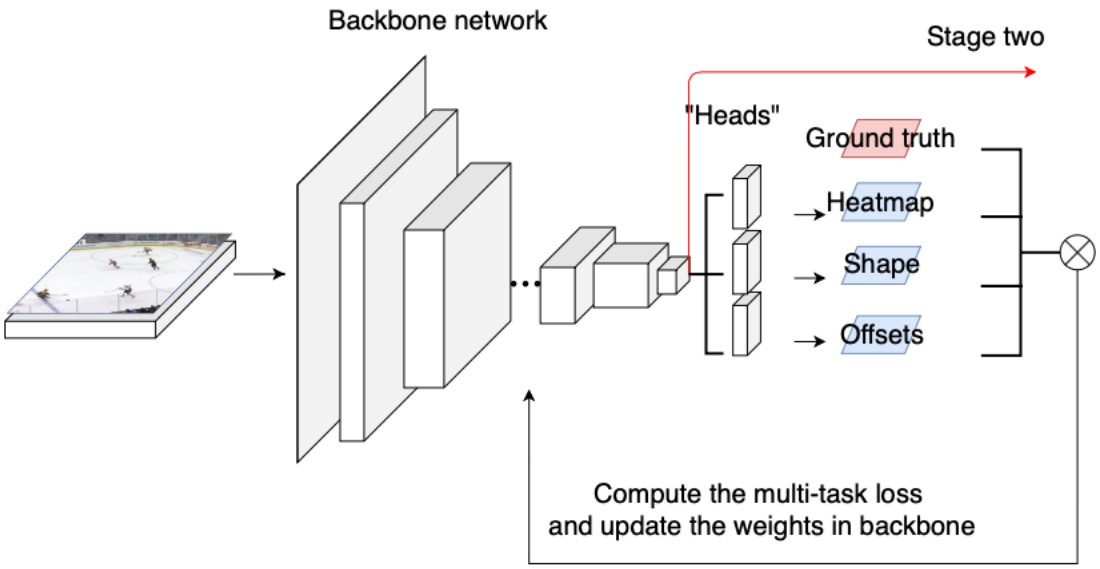

Figure 3.2.2.6 The structure of Stage One and the add-on used for training

To train the network, we consider Stage One as an individual detection system by adding three heads following the backbone network. These are used to predict the center points' heatmap, objects' shape, and center points' offsets. Each head uses the same feature map (shown in Figure 3.2.2.7) as input to predict its desired outcome. The three heads share a similar structure that contains two convolutional layers. Between convolutional layers, we add an activation function, ReLU, to complete the features' non-linear transformation. The transformed output is then sent to the next convolutional layer as the input. The kernels' sizes are 3 x 3 and 1 x 1 in these two convolutional layers. Since the padding parameter is "1" in the first layer and the stride is "1" in both layers, the resolution of the output remains consistent with the input. The only difference between the three heads is the number of the kernels in the second 1 x 1 convolutional layer. The dimensions of the final output in the three heads vary because of their different objectives. To predict the heatmap of center points, we need to calculate C heatmaps, where C is the number of object categories. In this project, C is equal to

1 because we are only concerned with the location of the puck. We need two variables, width and height, to depict appearance of the objects. The prediction of the offsets can be specified on the x-axis and y-axis, accordingly. Therefore, the numbers of output channels are 1, 2, 2 in three heads.

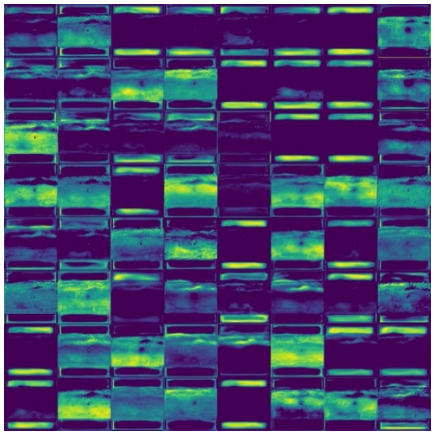

Figure 3.2.2.7 The visualized output feature maps (64 x 128 x 128) from the DLA network

3.2.2.3 Training Schemes

In order to train the backbone DLA network, we input the training set of the MPD into this network iteratively. We also compute the multi-task loss by comparing the differences between ground truth and predictions. The loss is then returned to the network to update the weights in the DLA network. By repeating these two processes, we can find the best performing model weights that will be further used in the following exploration. The associated loss of this system is defined as the combination of the loss from the three individual heads. We preprocess the ground truth by rescaling annotations to the same size as output feature maps. Then, we fine-tune Stage One and determine the best performing settings. After this exploration, we freeze the structure of Stage One and store the weights of the backbone network for the following exploration.

In this stage, we implement the DLA and three individual convolutional blocks as a whole system to predict the center point heatmap $\hat{Y}$, offsets $\hat{O}$, and shape $\tilde{S}$ for each frame. The system predicts a total of five values at each pixel. All of these three groups of heads share the

same backbone network. We assume that the center point of the bounding box is also the center point of the object. Let $I \in R^{W \times H \times 3}$ be an input frame (width = W and height = H). The center points' heatmap is denoted as $\hat{Y} \in [0,1]^{\frac{W}{R} \times \frac{H}{R} \times C}$, where R is the output stride and C is the number of object categories. In our case, C equals one since the puck is the only class in our dataset. The output stride downsamples the output prediction by a factor R. For example, if the original frame size is 512 x 512 and R = 4, the size of the output feature map will be 128 x 128. A prediction $\hat{Y}_{x,y,c} = 1$ corresponds to a detected center point, while $\hat{Y}_{x,y,c} = 0$ means that it is the background. For each original ground truth center point $p \in \mathcal{R}^2$, we plot a heatmap using a Gaussian kernel for each original ground truth center point, $p \in \mathcal{R}^2$. Firstly, we compute the equivalent coordinate of each center point, $\tilde{p} = \left\lfloor \frac{p}{R} \right\rfloor$, on a downsampled heatmap. Then we implement a Gaussian kernel to smooth the ground truth heatmap:

$$Y_{x,y,c} = \exp\left(\frac{(x-\tilde{p}_x)^2+(y-\tilde{p}_y)^2}{2\sigma_p^2}\right),$$

where σ_p is the object size of the adaptive standard deviation.

In the training stage, the overall loss function consists of three individual components:

$$L = L_{hm} + \lambda_{wh} L_{wh} + \lambda_{off} L_{off}$$

We do not normalize the scale and directly use the raw pixel coordinates. The loss is scaled by constant hyperparameters λ_{off} and λ_{size}. The recommended settings of these coefficients are $\lambda_{off} = 1$ and $\lambda_{size} = 0.1$. In the following discussion, we design a series of experiments to optimize the model by adjusting individual components' weights using a multi-loss function.

The training objective is a penalty-reduced pixel-wise logistic regression. We compute the loss from center point prediction with:

$$L_{hm} = \frac{1}{N} \sum_{x,y,c} \begin{cases} (1-\hat{Y}_{xyc})^{\alpha} \log\left(\hat{Y}_{xyc}\right) & if\ Y_{xyc} = 1 \\ (1-Y_{xyc})^{\beta} (\hat{Y}_{xyc})^{\alpha} \log\left(1-\hat{Y}_{xyc}\right) & otherwise, \end{cases}$$

where α and β are hyperparameters of the focal loss, and N is the number of center points in the input image. Referring to Law and Deng [42], we use $\alpha = 2$ and $\beta = 4$ in all experiments. A

discretization error occurs due to the downsampling stride. To eliminate the effect of this error, we introduce a two-dimension variable to refine the detection. The model is designed to make an additional offset prediction, $\hat{O} \in \mathcal{R}^{\frac{W}{R} \times \frac{H}{R} \times 2}$, for each pixel. The offsets are trained with an L1 loss:

$$L_{off} = \frac{1}{N} \sum_{p} \left| \tilde{O}_{\tilde{p}} - \left(\frac{p}{R} - \tilde{p} \right) \right|$$

Let $(x_k, y_k) \in \mathcal{R}^{\frac{W}{R} \times \frac{H}{R}}$ be the center point of the object k. The size of object k on the downsampled output is $S_k = (w_k, h_k)$. We use the convolutional network to predict the object's size: $\tilde{S} \in \mathcal{R}^{\frac{W}{R} \times \frac{H}{R} \times 2}$. To include the effect of object shape in tiny object detection, we use an L1 loss at each center point to compute L_{wh}:

$$L_{wh} = \frac{1}{N} \sum_{k=1}^{N} \left| S_k - \tilde{S}_{p_k} \right|$$

To summarize, we use a deep 2D convolutional network to extract the high-level representation of each still image (frame) in Stage One. The core component of this stage is a modified DLA network. Besides a basic 6-level DLA network, a significant number of iterative aggregation nodes are applied to compute and refine the final representation with the desired resolution. To train Stage One separately, we add three blocks to predict the center points' heatmap, objects' shape, and offsets. The loss functions for these three tasks are defined as follows. The total loss of Stage One is computed as a linear combination of three loss-functions, which are used to optimize the configurations and settings. The best performance configurations and network's weights are frozen for the later analyses.

3.2.3 Stage Two: Computing the Temporal Features

In Stage Two, we focus on leveraging the temporal information in video clips. The objective is to detect the puck in frame t = T. We preprocess the annotation file and generate the ground-truth center points' heatmap. In order to refine the frame's representation with

temporal information, we build a new dataset containing all the representations output from Stage One. We stack nine consecutive intermediate products (t = T – 4 to t = T + 4) as a new sample. Its corresponding ground truth is the preprocessed annotation of the frame t = T. Then, the stacked samples are input into a three-layer 3D convolutional network that enables the final representation to consist of the temporal features as well. After extracting the ultimate representation, three heads are added to compute the center points' heatmap and corresponding shape and offsets. These three convolutional blocks have the same structures as the ones in Stage One. Therefore, transferring the weights from frozen Stage One accelerates and improves the training in this stage. Eventually, the final detection result is computed by merging the output of the three heads. More details about Stage Two are presented in the following section.

3.2.3.1 3D convolution

As we introduced above, the emergence of convolutional neural networks led to the breakthrough of many computer vision tasks. The convolutional layer is the most important component in CNNs, which has the property to preserve the spatial relationships between input data points. Each convolutional layer consists of a certain number of fixed-size kernels. These kernels are applied over the input data and produce a weighted sum as the output. Convolutional layers can be seen as special filters with learnable parameters used to extract features from input data.

Various modified convolutional layers have been proposed to meet the requirements of different applications. The most commonly used convolutional layer involves 2-dimension convolutional computation. The dimension of the convolutional layer is determined by the number of axes kernels sliding along. As shown in Figure 3.2.3.1 (a), in 2D convolutional layers, kernels are applied as sliding windows moving along the x-axis and y-axis to output 2D features. For example, processing an RGB image only needs a 2D convolutional layer. Both of the input data and each convolutional kernel have three channels. However, each kernel can only compute one 2D output. The depth of the production of the whole convolutional layer is the

number of kernels used. When it comes to video processing, the video data require one additional dimension than the image data: that is, the temporal dimension. The 3D convolutional layer is proposed to tackle the challenge of video feature extraction. Figure 3.2.3.1 (b) shows a simple demo of a 3D convolutional operation. This new 3D convolution follows almost all the procedures in the 2D convolution. The only difference is that one more channel has been added to the kernel to process the temporal dimension. Also, the number of sliding directions have been increased to three. With the help of 3D convolutional layers, we can adequately leverage temporal information in the videos. The drawback of the 3D convolutional layer is that it consumes massive computational resources. Thus, we do not directly implement 3D convolutional processing. Instead, we use the powerful 2D CNN as a feature engine to compute each frame's representation and then connect them with a 3D CNN to further refine the representation (Discussed below.).

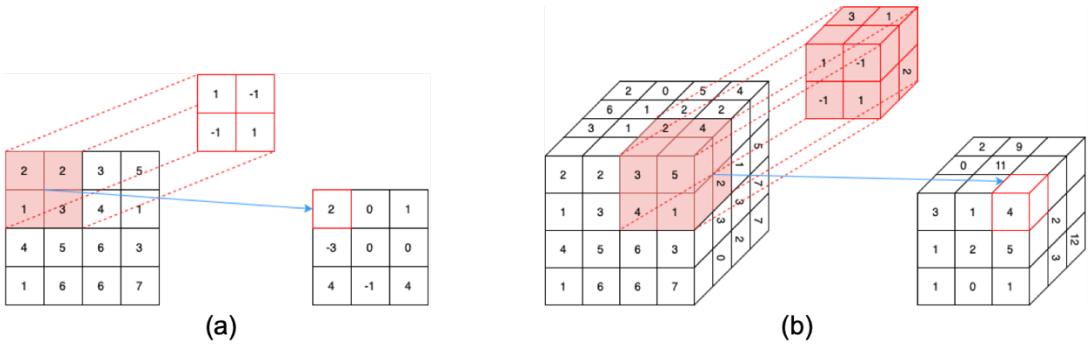

Figure 3.2.3.1 Demos of the convolution operation in 2D (a) and 3D (b) convolutional layers

3.2.3.2 Structure

The input to Stage Two is a series of representations output from Stage One. The idea is to leverage the temporal information embedded in a video sequence to refine the puck detection in each frame. The structure of this stage is shown in Figure 3.2.3.2. We first re-organize the intermediate products from Stage One as the new training data. For data from a specific time point, we stack its feature map (64 x 128 x 128) with the other eight feature maps computed from its contiguous frames. Thus, the resolution of new data at t = T is:

$$Input = (Chennel_{in} = 64, D_{in} = 9, H_{in} = 128, W_{in} = 128)$$

Then, these four-dimensional data are input into a three-layer 3D CNN. The kernel sizes are (3, 3, 3), (5, 3, 3), (3, 3, 3) in each layer. The padding parameter is 0 at the temporal dimension, while it is equal to 1 at width-dimension and height-dimension (padding = (1, 1, 0)). The convolutional computation changes the shape of the data following these rules:

$$Chennel_{out} = \#\ of\ filters$$

$$D_{out} = \left\lfloor \frac{D_{in} + 2 \times padding[0] - kernel\ size[0]}{stride[0]} + 1 \right\rfloor$$

$$H_{out} = \left\lfloor \frac{H_{in} + 2 \times padding[1] - kernel\ size[1]}{stride[1]} + 1 \right\rfloor$$

$$W_{out} = \left\lfloor \frac{W_{in} + 2 \times padding[2] - kernel\ size[2]}{stride[2]} + 1 \right\rfloor$$

Giving stride = (1, 1, 1), the resolution of the output is:

$$Output = (Chennel_{out} = 64, D_{in} = 1, H_{in} = 128, W_{in} = 128)$$

After that, the heads are applied sequentially to compute the center points' heatmap, corresponding shape and offsets. Since these three heads have the same structure as those in stage one, we can load the trained weights of these three blocks from Stage One as the new pre-trained weights to accelerate the training process. Finally, the refined prediction of the puck is made by combining the outputs from these three heads.

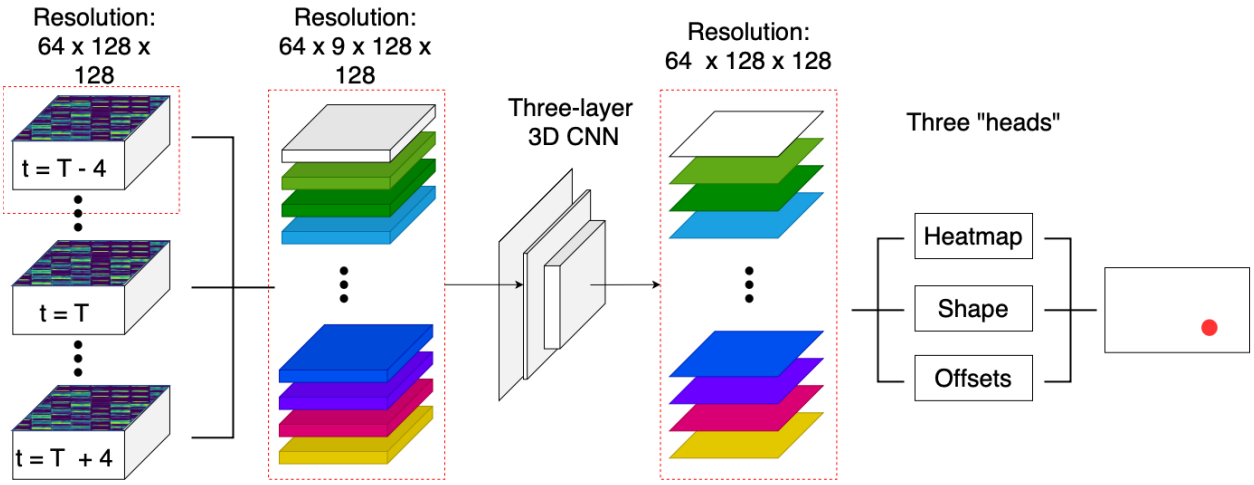

Figure 3.2.3.2 The structure of Stage Two

3.3 Evaluation Metrics

Designing suitable evaluation metrics is essential because it allows us to compare the new model with other state-of-art models in the field of object detection. With the help of statistical analysis, we can intuitively understand the model's strength and weakness. Additionally, reasonable criteria can help to select better hyperparameters to optimize the model. Thus, one of the objectives of this project is to find an evaluation method for the tiny object detection.

Before globally evaluating the quality of the models, we need to determine each prediction's correctness. There is no ready-to-go evaluation method for tiny object detection. The most commonly-used metric is the Intersection of Union (IoU). Calculating the IoU is as simple as dividing the area of overlap between predicted and ground-truth bounding boxes by the area of their union. The IoU measures how much the prediction correctly matches the corresponding annotation. For example, IoU = 1 indicates that the prediction perfectly matches the ground truth. In contrast, prediction with IoU = 0 is seen as a wrong proposal. By giving an IoU threshold in advance, all predictions having larger IoU scores are considered correct. The drawback of implementing the IoU is obvious. With this criterion, the prediction of the object's shape has a more significant impact on the final decision. When it comes to tiny object detection, the prediction of the object's feature point (center point) instead of the object's shape is given a higher priority. Brutally using IoU to discriminate predictions is too coarse to reflect the model's true capabilities accurately.

Therefore, we develop a custom evaluation method for the puck detection, which can also be applied to other applications in the domain of tiny object detection. The whole evaluating procedure can be divided into two steps. Firstly, we need to decide whether each prediction matches the corresponding ground truth. Specific criteria are applied to determine the correctness of each detected result. Secondly, further statistical analysis is conducted to reveal the models' performance globally. In this step, several metrics are used to show the

capabilities of the models in different aspects.

We assume that the center point of the annotated bounding box is also the center point of the object. The ground-truth and predicted center points are denoted as $P(x, y)$ and $P'(x', y')$, respectively. Instead of computing the IoU between bounding boxes, we calculate the Euclidean distance between P and P'. Euclidean distance, also known as the L2 distance, is defined as the shortest distance between two points in N-dimensional space (two dimensions in this case). If the distance is less than the selected distance threshold, the prediction is considered correct. The L2 distance between them can be computed as:

$$Distance_{L2} = \sqrt[2]{(x - x')^2 + (y - y')^2}$$

To enhance the quality of the election, the heatmap indicates the possibility of each pixel being the center point of an object. We generate a prediction at each pixel on the heatmap by retrieving its corresponding width, height, and offsets. The "possibility score" also represents the confidence of each proposal, which can be further used to eliminate most of the irrelevant predictions. Initially, we have 128 x 128 predictions in each frame. To reduce unnecessary computation, the model sorts all predictions in the descending order of confidence score. Only top-100 proposals are kept for the following prediction. We further filter 100 predictions with a confidence threshold. Although the number of predictions drops significantly at this stage, there remain redundant predictions with qualifying confidence scores. Rather than detecting an object once, the model might generate several similar predictions from a cluster of center points. As shown in Figure 3.3.1 (a), there may exist more than one satisfactory prediction around one ground truth. Non-Maximum Suppression (NMS) is an algorithm to get rid of these unwanted predictions of the same object (shown in Figure 3.3.1 (b)).

Figure 3.3.1 Comparison between predictions before (a) and after (b) the use of NMS

To explicitly explain the operations behind NMS, we denote the set of pre-selected predictions as B. The final proposal list is denoted as D, which is initially empty. Firstly, the proposal with the highest confidence score is found and moved from B to D. Secondly, we calculate the L2 distance between the selected proposal and the remaining proposals in B. For all the remaining proposals in B, If the L2 distance between them and the selected proposal is smaller than the NMS threshold, they will be eliminated from B. Thirdly, we repeat previous steps, where a new-highest proposal is selected into D, and the unqualified ones are removed from B. Finally, the computational loop ends when no element remains in B. The output of NMS is D which contains all unique predictions with high confidence scores.

We note that in the field of machine learning, the confusion matrix is widely used to describe the performance of a model on a set of annotated data. It explicitly tells what the model does that is right and wrong. The general idea is to count the number of times instances of each category are classified correctly. The rows of the confusion matrix correspond to what the has model predicted, while the columns correspond to the ground truth. As shown in Figure 3.3.2, there are four cells in a basic confusion matrix: True Positive, True Negative, False Positive, False Negative. To understand these terms more intuitively, we use puck detection as an example. If the model correctly predicts the location of a puck, the prediction will be counted as a True Positive. When the model wrongly detects other objects or background as the puck, the prediction belongs to False Positive. The prediction will be considered as False Negative if

the model doesn't successfully detect the puck where there exists one.

		Ground Truth	
		0	1
Prediction	0	True Negative	False Negative
	1	False Positive	True Positive

Figure 3.3.2 Confusion matrix

In order to conduct further statistical analysis, we employ all the predictions to create a confusion matrix. Once it is obtained, we implement more useful metrics to show the model's capabilities. The two most fundamental metrics are Precision and Recall. Precision is the percentage of accurate predictions. While Recall, also known as sensitivity, depicts how many out of all annotated samples have been correctly predicted. Precision and Recall can be denoted as:

$$Precision = \frac{True\ Positive}{True\ Positive + False\ Positive}$$

$$Recall = \frac{True\ Positive}{True\ Positive + False\ Negative}$$

The overall performance of the model can be measured in various ways. The priority of different applications can vary as well. For example, spam email classification allows the model to miss some spam emails but not to make the wrong prediction. In the other hand, missile detection requires the model to find the missile even if that means more wrong predictions. Precision is a more crucial criterion for evaluating a spam email classifier, while Recall is more reasonable for the evaluation of a missile detector. The trade-off between these two demands needs to be explored. Higher Precision means that the model is less likely to detect irrelevant background and objects as the puck. In contrast, higher Recall means that less ground-truth samples are missed. Both of these are essential criteria when it comes to model evaluation. However, the model with higher Recall tends to have lower Precision, and vice versa. If the

model recalls everything, it must keep generating predictions which are not all accurate, hence it lowers the overall Precision. We can choose either of them as the criterion to evaluate the model. As for puck detection, neither Precision nor Recall should be given a higher priority. Thus, we apply a new metric which takes both Precision and Recall into account and enables us to explore a better trade-off. This metric is known as the F1 score, which calculates the harmonic mean of Precision and Recall. This metric is much simpler to work with, as we only need to maximize one score, rather than balancing two separate scores. The F1 score is denoted as:

$$F1 = 2 \times \frac{Precision \times Recall}{Presision + Recall}$$

To summarize, the new evaluation method is very task-specific, which works well on tiny object detection. It efficiently serves the puck detection application because it computes the L2 distance between feature points rather than IoU between bounding boxes. L2 distance is the critical criterion to determine the correctness of each prediction. Since the model makes a significant number of predictions for each frame, we filter all proposals with different thresholds and Non-Maximum Suppression. We employ all prediction results into a confusion matrix to globally measure the capabilities of the model. After that, we implement several evaluation metrics, such as Precision, Recall, F1 score, and Precision-Recall curve. The strength and weakness of the model can then be intuitively and comprehensively revealed.

Chapter 4 Experiments

In Chapter 4, we explicitly explain the processes of selecting the appropriate settings, designing the network architectures, fine-tuning training hyper-parameters, and comparing between our model with the state-of-art object detectors. The detailed training information is listed in 4.1, including most optimizing strategies and hyper-parameters. This section also covers the information of the hardware used to training and inferencing models, helping other researchers reproduce our work on their servers. In 4.2, we visualize the outputs from both training and inference. By analyzing the trends of multi-task losses and the model's performance, we determine the threshold for the number of training iterations that prevent models from being overfitted. Through this series of experiments, we can also tell the different learning abilities of three sub-tasks. The backbone network in Stage One is used as a feature engine that can extract summarized representations for each frame. The model's accuracy is strongly related to these representations' qualities, while the feature engine has a tremendous impact on representation extraction. Thus, different state-of-art backbone networks are validated and compared in 4.3. We thoroughly explore the most productive loss function for both stages in 4.4. Since multi-task loss is a linear combination of three sub-task losses: center points' heatmap, shapes, and offsets, we optimize the coefficients before each term to maximize the accuracy of tiny object detection. In 4.5, we conduct a series of experiments to explore the relationships between the input data's resolution and the model's performance in different aspects. We summarize all the experiments in 4.6.

4.1 Implementation Details

In this project, NVIDIA TITAN RTX [43] takes care of both model training and inference. The server provides 24 GB memory running at Gigabits per second for up to 672 GB/s of memory bandwidth, enabling us to train large machine learning models within an acceptable time. All

the experiments related to running time comparison are conducted on the same server and with identical conditions. The default learning rate is 6.26e-5 for batch size 32 on a single GPU. According to the linear learning rate rule, the learning rate should be decided by the number of available GPUs. For example, using distributed data parallel- training on two GPUs, the learning rate doubles to 1.25e-4.

To evaluate the tiny and fast-moving object detector's performance, we train the prototype with the McGill Puck Dataset. The dataset is initially divided into training, validation, and test set in a ratio of 8:1:1. We reserve the test set to the last to prevent potential overfitting in the fine-tuning process. The results in the following section that are shown without specific labeling, indicate the performance on the validation set. Also, all the hyperparameters and the model's architecture are selected based on the validation results. In the end, when we examine the ultimate model on the test set, and the results are in our favor, they show no evidence of over-learning.

The evaluation metrics are considerably different from the ones used in normal-size object detection. First, instead of the IoU, the L2 distance between the center point of the ground-truth bounding box and predicted center point is the criterion to determine the correctness of the individual predictions. Standard machine learning evaluation metrics, Precision, Recall, and F1 Score, depict the model's overall performance, which requires a pre-setting distance threshold. In the following research, DisThre = 2.5 is pre-set in the evaluating process. Since the model is sensitive to the choice of the distance threshold, we compute the model's F1 score in the range given by DisThre = [0.5, 3.5]. The relative area under that curve (AUC score) convincingly illustrates the overall performance of the model.

4.2 Training and Inference

4.2.1 Training

The whole training processing can be divided into two separate stages. In both phases, we use the same multi-task loss function. The total number of learnable parameters in Stage One is much larger than Stage Two, so we train Stage One more carefully (more training epochs). In the following section, we present experiments exploring the training procedure, which can help determine better training hyperparameters and settings.

The curse of dimensionality, also known as overfitting, is a typical modeling error when a function is too closely fit to a limited amount of data points. The objective of a supervised learning model is to learn the pattern from labeled data and generalize from the training data to unseen data. Underfitted models fail to sufficiently understand the underlying pattern and perform poorly on both the training and validation sets. In contrast, a model that learns the pattern in the training set too well but doesn't perform equally well on the validation set is considered overfitting. With the occurrences of extremely deep and complicated models, overfitting has become a headache for all researchers. Checking and preventing overfitting is an inevitable procedure in the field of machine learning.

One of the most efficient ways to avoid this issue is known as early stopping. We can stop teaching the model with the same training set before it loses the ability to generalize. In this project, we set the maximum number of training iterations to 140 and validate the training models every five epochs. Figure 4.2.1 shows the tendencies of multi-task loss in both training and validation.

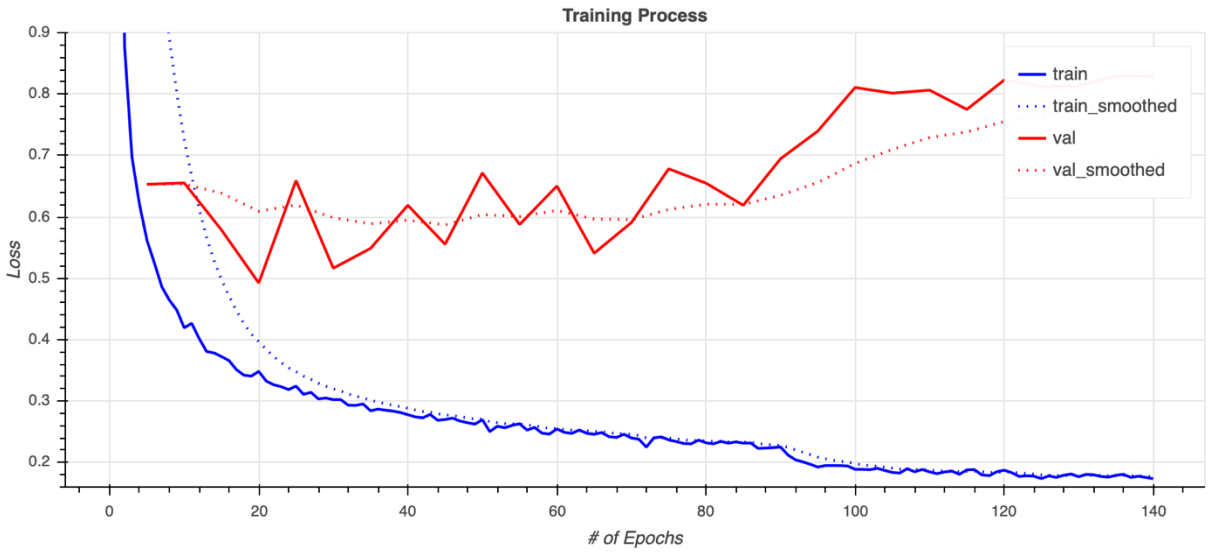

Figure 4.2.1 The tendencies of multi-task loss in both training and validation

The dotted lines in the figure are the EMA (Exponential Moving Average) smoothed curves that show the trends in the loss during training and validation procedures. EMA, a calculation to analyze data points by creating a series of averages of different subsets of the full data set, is widely used to smooth the random fluctuations to provide a more intuitive observation. EMA can be denoted as:

$$S_t = \begin{cases} Y_1, & if\ t = 1 \\ \alpha Y_t + (1-\alpha)S_{t-1}, & otherwise \end{cases}$$

, where the coefficient α represents the weight given to the tth sample in the original list. S_t and Y_t are the actual and smoothed values at the tth step, respectively. The α used in this experiment equals 0.8.

Analyzing the multi-task loss trend in Figure 1, we notice that the training loss gradually converges to 0.18 as the training epochs approach 140. The validation loss slowly converges to 0.6 starting from the 40th epoch, and slightly fluctuates around 0.6 between the 40th and the 70th epoch. However, the validation loss begins to rebound from the 70th epoch. According to these trends, we suspect that overfitting occurs after the 70th epoch. Nevertheless, there still could be many other reasons that can scientifically explain this result. For example, since the loss consists of three individual task-specific components, differences between learning speed may exist. These can influence the final training in a significant way. In order to find the most

accurate overfitting threshold, we save the three components. The task-specific losses (heatmap, shape, and offsets) in training and validating procedures are recorded in Figure 4.2.2, 4.2.3, and 4.2.4, respectively.

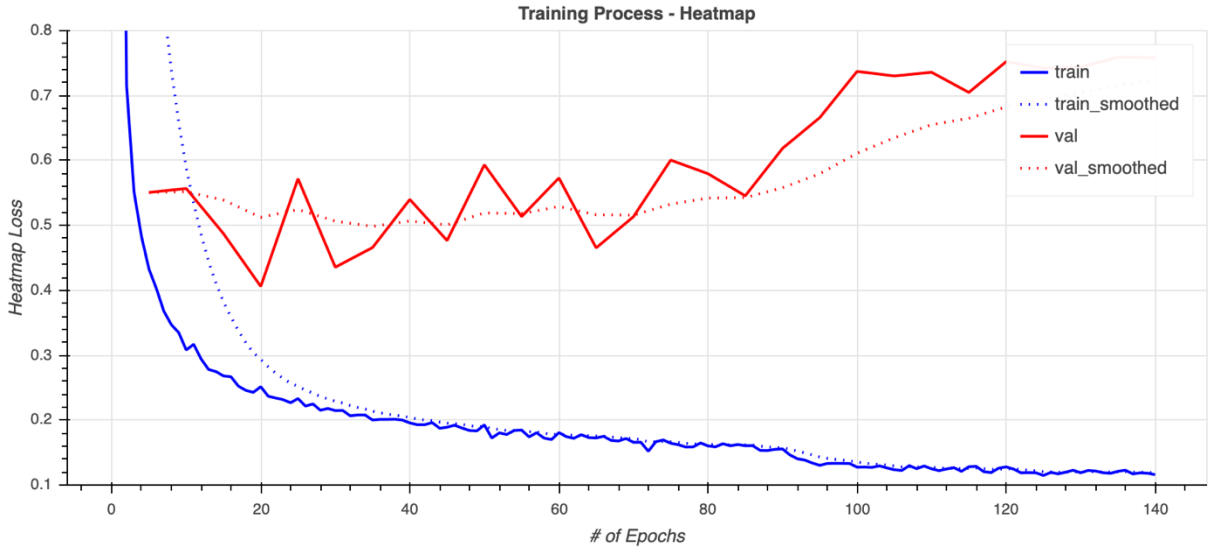

Figure 4.2.2 The tendencies of heatmap loss in both training and validation

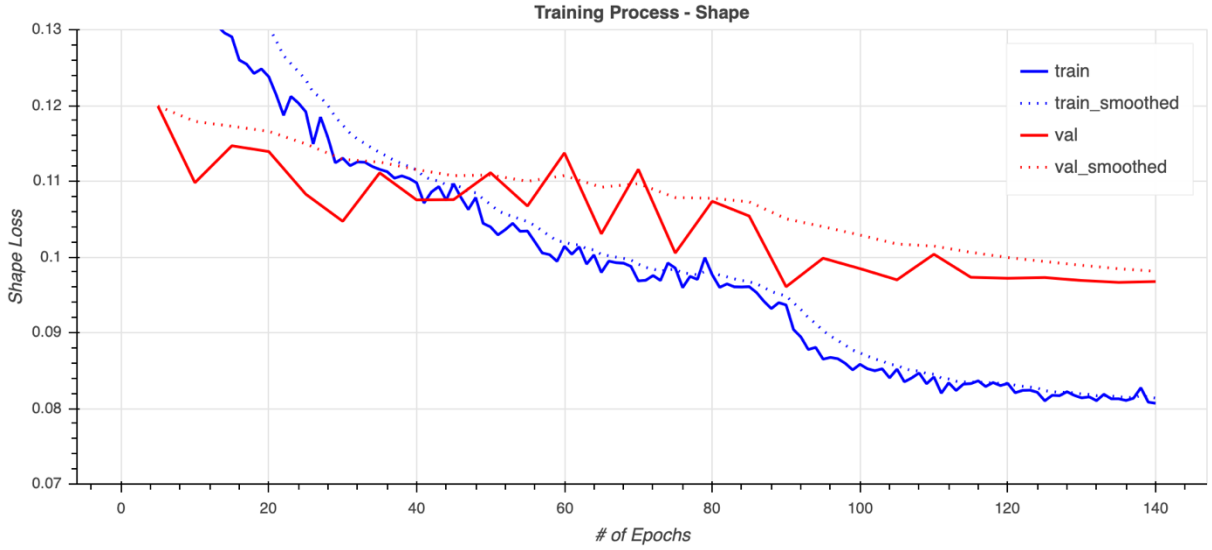

Figure 4.2.3 The tendencies of shape loss in both training and validation

Figure 4.2.4 The tendencies of the offset loss in both training and validation

We note that the trends of the heatmap loss highly parallel the ones of the whole multi-task loss. Overfitting also occurs at around the 70th epoch. Unlike heatmap loss, both shape and offset losses have perfect learning curves, showing no overfitting evidence. By comparing the number of epochs the three tasks require to converge, we notice that the "heatmap" is a much faster learner compared to the "shape" and "offsets". Also, the loss values computed from heatmap prediction are much higher than the rest. The final loss is a linear combination of these three losses. We conclude that the task of heatmap prediction is likely to be overfitting after training 70 epochs.

Besides the comparisons of loss, we implement evaluation metrics at each validation step in order to determine whether to model starts to lose the ability of generalization. These metrics can explicitly tell when the model is over-trained. Figure 4.2.5, 4.2.6, 4.2.7 depict the model's performance at each epoch in terms of Precision, Recall, and F1 Score given the constant distance threshold: DisThre = 2.5. The curve of Precision starts with drastic fluctuation and a steady increase after the 70th epoch. The Recall is significantly improved before the 70th epoch and shows a slight decrease after the 80th epoch. Regardless of some small waves, the F1 score retains an upward trend from the start to the end.

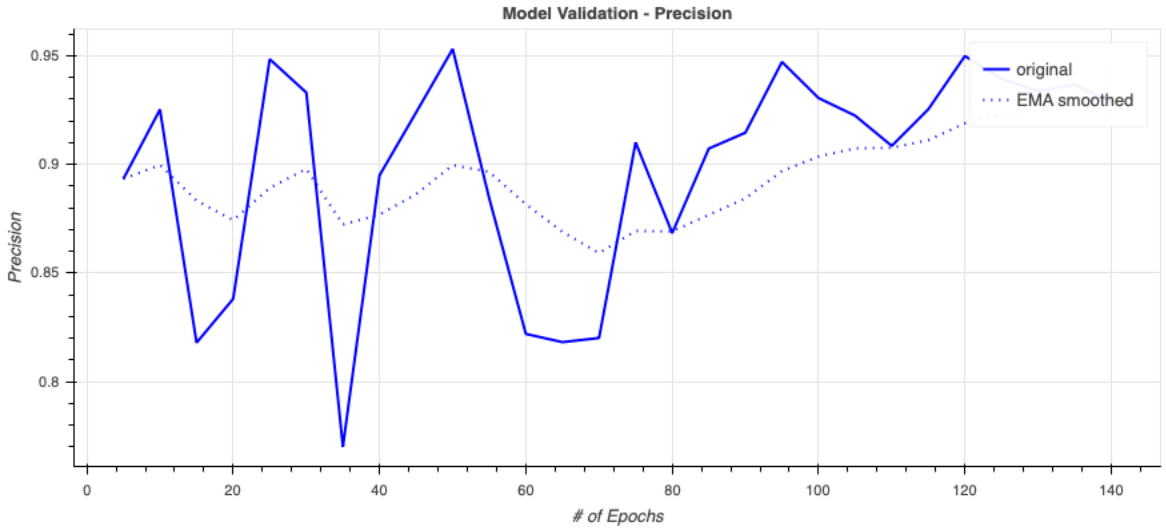

Figure 4.2.5 Evaluating the validation model in terms of the Precision

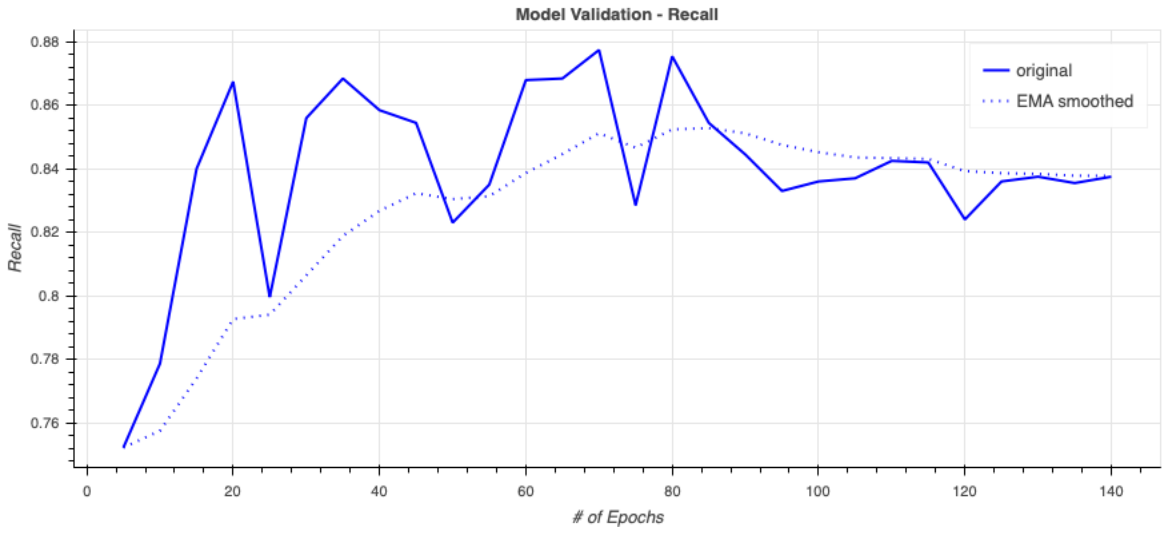

Figure 4.2.6 Evaluating the validation model in terms of the Recall

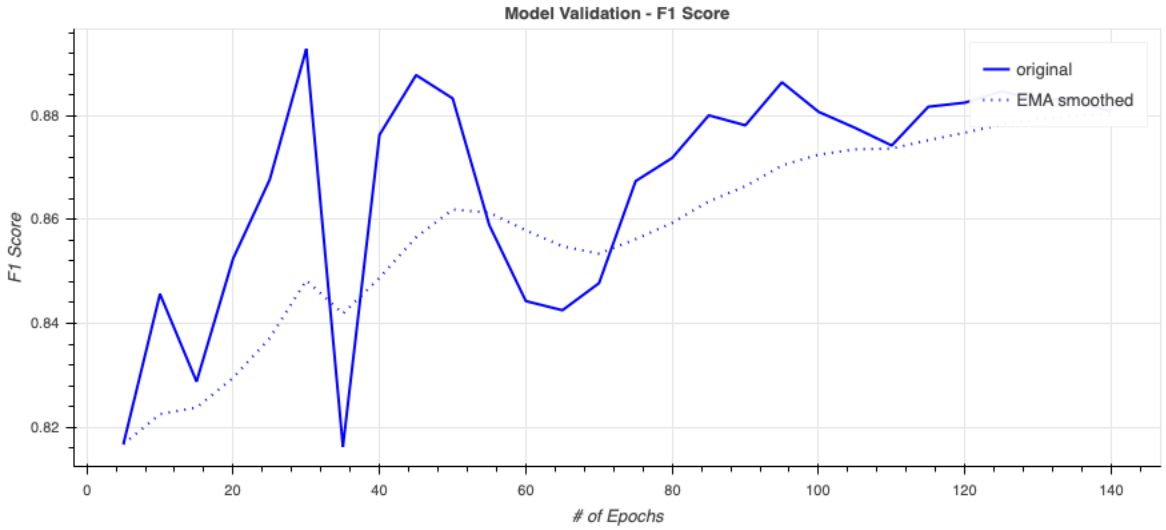

Figure 4.2.7 Evaluating the validation model in terms of the F1 Score

The previous results are measured using a specific distance threshold that might lead to unexpected fluctuations. In order to average the distance threshold effect, we define the AUC score and the relative area under the F1 score curve giving $DisThre \in [0.5, 3]$. As shown in Figure 4.2.8, the AUC score trend mostly matches the F1 score at $DisThre = 2.5$. As the number of epochs increase, the AUC score approaches 0.75, and the performance keeps steady after the 120th epoch.

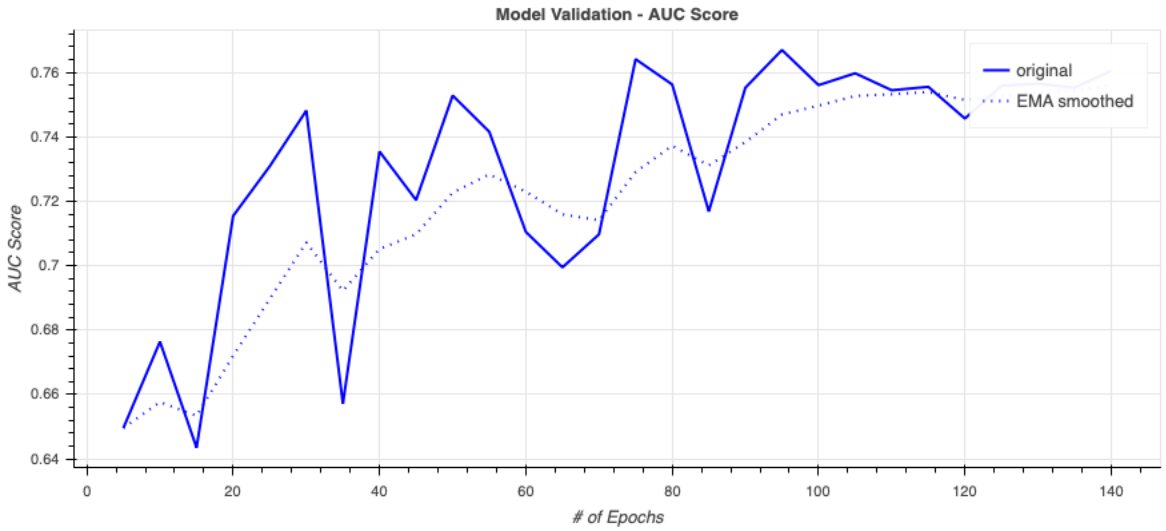

Figure 4.2.8 Evaluating the validation model in terms of AUC score

In summary, we hypothesize that the model will not be overfitting before the 140th epoch. There is strong evidence showing that heatmap prediction can be trained much faster

than the other two, leading to moderate overfitting. In the future, this problem、 can be solved by dividing the loss function into two intervals. We can gradually decrease the coefficient before heatmap loss, enabling the model to focus on learning information related to an object's shape and offsets predictions.

4.2.2 Inference

When given a new unseen video, our model can make accurate predictions of the puck's existence and location. The detection is conducted on each frame and needs to leverage the information from four frames before and after. For example, to detect the puck in frame t = 10, we firstly organize frames from t = 6 to t = 14 as a new cluster. Then, in Stage One, a pre-trained 2D CNN processes each frame and outputs the high-dimensional representation (64 x 128 x 128). The intermediate products are stacked together as the new input (64 x 9 x 128 x 128) for Stage Two. With the help of a 3-layer 3D CNN, a more summarized representation (64 x 1 x 128 x 128) containing temporal information of frame t = 10 is output from Stage Two. Three individual convolutional blocks predict the center points' heatmap and the corresponding shape and offsets. Finally, the ultimate predictions are retrieved and returned to the original frame. We set the downsampling ratio to 4 and the initial resolution to 512. Theoretically, we have 128 x 128 proposals for each frame. However, most of them are incorrect predictions with extremely low confidence scores. To concentrate on the confident predictions, we create procedures to filter all predictions and eliminate unnecessary ones. In order to intuitively present the post-processing, we visualize the results after each step. Table 4.2.1 shows the number of forecasts in each stage.

Inference Steps	Initial Stage	Top-100 Predictions	Confidence Threshold	NMS Processing
# of predictions	16,384	100	5	4

Table 4.2.1 the number of remaining predictions in each post-processing steps

The initial number of predictions equals 16,384 because we can compute one proposal. The final center point heatmap has C x 128 x 128 pixels, where C = 1. The value of each pixel represents the possibility of that pixel being the center point of an object and measures the corresponding prediction's confidence. Moreover, there are corresponding shape and offset values for every pixel in the heatmap. Since we can generate a prediction at each pixel, the initial number of predictions is 16,384. Most of the proposals at this stage are incorrect. Firstly, we seek the top-100 predictions in all categories based on their confidence scores. In this project, our focus is concentrated on the puck. Thus, the output contains the top-100 predictions of the puck. Given a constant confidence threshold, ScoreThre = 0.1, can further decrease the number of predictions to an acceptable level. However, the filtered results still contain redundant predictions around the ground truth. Thus, we implement Non-Maximum Suppression to eliminate overlapping predictions and keep the most confident ones within a specific range (NMSThre = 5). The predictions are shown in Figure 4.2.11.

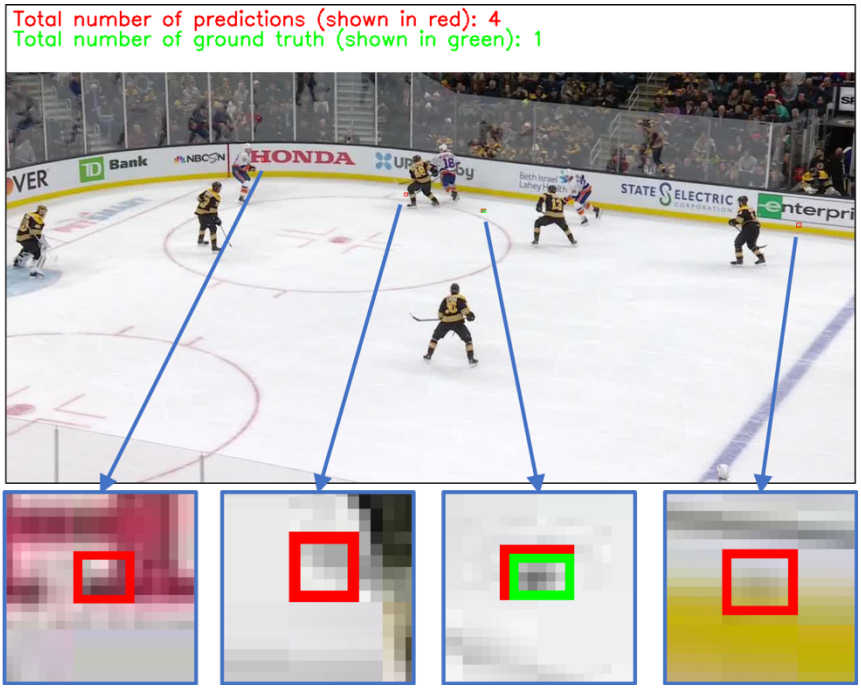

Figure 4.2.11 Final predictions (ScoreThre = 0.1, NMSThre = 5). The green and red bounding boxes show the ground-truth and predicted locations of the puck in this frame. In order to

closely observe the results, we zoom in on the regions that contain either the predicted or ground-truth puck (shown in blue windows under the original frame).

Clearly, some additional processing is still required. By analyzing the results, we note that the size, also known as the confidence score, of the correct prediction is much larger than other incorrect predictions. Therefore, we can increase the confidence threshold in order to eliminate these wrong predictions. Figure 4.2.12 shows the final output, given ScoreThre = 0.2 and NMSThre = 5.

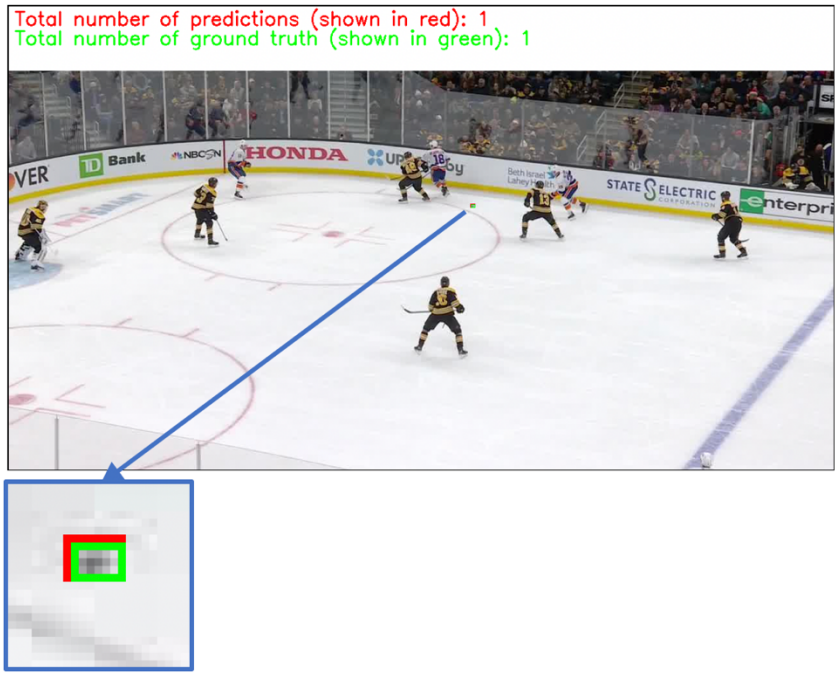

Figure 4.2.12 Final predictions (ScoreThre = 0.2, NMSThre = 5). The green and red bounding boxes show the ground-truth and predicted locations of the puck in this frame. In order to closely observe the results, we zoom in on the regions that contain either the predicted or ground-truth puck (shown in blue windows under the original frame).

4.3 Backbones

In this project, we implement the so-called “backbone + multi heads” structure [9] in order to precisely locate the tiny object. We require the backbone network to summarize both

the semantic and appearance information. The particular selection of the backbone network significantly impacts the model's final performance because it determines the quality of the extracted representations and precisely locates the tiny objects. As introduced in Chapter 3, the backbone network that is used to extract the representation for each frame is a Deep Layer Aggregation network plus bidirectional iterative aggregations. This architecture can equally fuse the output from the shallow to deep layers. Besides that, the bidirectional iterative aggregations enable the model to output the desired-size feature maps, which can conserve the detailed information as much as possible.

The ResNet network [37] was a pioneering model involving the idea of feature fusion. It was proposed in 2015 and drew a massive amount of attention because of its extremely outstanding performance. ResNet was originally motivated by the need to deal with the notorious vanishing gradient problem caused by the need to search networks with ever increasing depth. Consequently, as the network got deeper and deeper, its performance saturated or even started degrading rapidly. ResNet's core idea introduced a so-called "identity shortcut connection" that skipped one or more layers, which can be seen as a small aggregation between layers. Even though ResNet was invented many years ago, it is still the first choice for many machine learning applications. In this experiment, we replace the Deep Layer Aggregation network with the popular ResNet18 and ResNet50. The Models' performance is shown in Table 4.3.1.

	Precision	Recall	F1 Score	AUC Score	Inference Time (s)
Res101	0.374	0.817	0.514	62.437	0.596
Res18	0.406	0.816	0.543	63.324	0.326
DLA	0.908	0.867	0.887	76.493	2.7289

Table 4.3.1 The comparison between different backbone networks

The average inference speed of the model with ResNet is much higher than the model that uses a DLA network. However, the model using DLA as the backbone shows significant

advantages in the aspect of prediction accuracy. There is a reasonable explanation for these differences. In DLA, we implement a bi-directional iterative aggregation to upsample the features to the desired resolution. In contrast, for ResNet18 and ResNet50 (the architectures are shown in Table 4.3.2), we only implement three deconvolutional layers to enlarge the feature maps. The complicated upsampling layers can efficiently improve the model's accuracy for tiny object detection at the cost of longer processing time.

Layer Name	Downsampling Ratio	ReNet18	ResNet50
Conv1	2	7 x 7, 64, stride 2	
Conv2_x	4	3 x 3 max pool, stride 2	
		$\begin{cases}3\times3, 64\\ 3\times3, 64\end{cases}$ x 2	$\begin{cases}1\times1, 64\\ 3\times3, 64\\ 1\times1, 256\end{cases}$ x 3
Conv3_x	8	$\begin{cases}3\times3, 128\\ 3\times3, 128\end{cases}$ x 2	$\begin{cases}1\times1, 128\\ 3\times3, 128\\ 1\times1, 512\end{cases}$ x 3
Conv4_x	16	$\begin{cases}3\times3, 256\\ 3\times3, 256\end{cases}$ x 2	$\begin{cases}1\times1, 256\\ 3\times3, 256\\ 1\times1, 1024\end{cases}$ x 3
Conv5_x	32	$\begin{cases}3\times3, 512\\ 3\times3, 512\end{cases}$ x 2	$\begin{cases}1\times1, 512\\ 3\times3, 512\\ 1\times1, 2048\end{cases}$ x 3

Table 4.3.2 The architectures of ResNet18 and ResNet50

4.4 Multi-task Loss

As introduced above, the loss function used for adjusting the weights in both Stage One and Stage Two is a linear combination of three individual task-specific losses: $L = \lambda_{hm}L_{hm} + \lambda_{wh}L_{wh} + \lambda_{off}L_{off}$. All the learnable parameters in the model are automatically updated according to the loss values in the training process. In other words, the loss function controls

the model's learning direction. Different coefficients give different-level priorities to the three tasks, which can change the model's final performance in an unexpected way. Thus, we need to explore the most effective form of the loss function. We set $\lambda_{hm} = 1$ to simplify the whole optimizing process, which leaves only two hyper-parameters to determine. Except for λ_{wh} and λ_{off}, all hyper-parameters and settings are frozen in this series of experiments. The exploration aims to find the most effective form of loss function for tiny object detection.

Even though only two hyper-parameters need to be optimized, it is too time-consuming to examine all of the possible combinations. Therefore, we first optimize the two parameters separately and then refine them to seek global optimal solutions. Simply speaking, we disable one variable λ_{wh} and try different values for the other variable λ_{off}. The value that gives the best performance on the validation set is the local optimal solution for λ_{off}. By repeating this procedure, we can also obtain the local optimal solution for λ_{wh}. After that, we set new ranges for the two variables based on the previous stage results. Finally, we involve λ_{wh} and λ_{off} simultaneously and assign their values from the optimal ranges. The values leading to the best performance are considered as the most appropriate and functional settings for λ_{wh} and λ_{off}.

To determine the most productive range of λ_{off}, we set λ_{wh} to zero. A series of models are trained with the loss function: $L = L_{hm} + \lambda_{off} L_{off}$ where $\lambda_{off} \in [0, 0.1, 0.2, 0.5, 1, 2, 5, 10]$. After that, we evaluate all models using the same validation set. The results of all experiments are shown in Table 4.4.1, where the outstanding results are in red. Precision, Recall, and F1 score are measured under a particular distance threshold (DisThre = 2.5). AUC score is the relative area under the curve of the F1 Score given DisThre in the range of [0.5, 3].

	λ_{wh}	λ_{off}	Precision	Recall	F1	AUC Score
Exp0001		10	0.492	0.853	0.624	74.172
Exp0002	0	5	0.35	0.829	0.493	72.642
Exp0003		2	0.329	0.851	0.474	74.167

Exp0004	1	0.340	0.857	0.487	74.679
Exp0005	0.5	0.273	0.841	0.412	72.981
Exp0006	0.2	0.369	0.843	0.513	71.104
Exp0007	0.1	0.255	0.854	0.393	69.493
Exp0008	0	0.169	0.551	0.259	11.564

Table 4.4.1 The performance of models trained with different values of λ_{off}

The model trained with $\lambda_{off} = 10$ has the highest Precision and F1 Score, giving DisThre = 2.5. However, when comparing the models' performance with the AUC and Recall, the model trained with $\lambda_{off} = \lambda_{hm} = 1$ slightly leads the others. Note that setting $\lambda_{off} = 0$ decreases the model's capability in all aspects.

Figure 4.4.1 depicts the relationship between the values of λ_{off} and the models' AUC scores. The peak occurs at $\lambda_{off} = 1$, and the second peak, very close to the former one, is at $\lambda_{off} = 10$. Theoretically analyzing, the prediction of the center points' offsets can boost the model's performance, especially for tiny object detection. When we retrieve the predictions back to original image, each pixel in the heatmap is assumed to be an anchor. However, in the original input, the center point of the object may not be exactly at these anchors. The distance between the ground-truth center point and predicted anchor is the deviation needs to be considered. The effect of this deviation may not influence the accuracy of normal-size object detection. However, when it comes to tiny object detection, the model requires much higher precision. This deviation will significantly drag down the model's performance. The results clearly verify this assumption because the accuracy of the model decreases rapidly, given a smaller λ_{off}. Also, the positive influence of λ_{off} seems to reach a ceiling after a certain threshold. After analyzing this series of experiments, we take $\lambda_{off} = 1$ as the optimal solution for two reasons: 1. The metric evaluating the overall performance, the AUC score, indicates that $\lambda_{off} = 1$ is a better choice. 2. Technically, the prediction of offsets for each center point plays an essential role in tiny object detection.

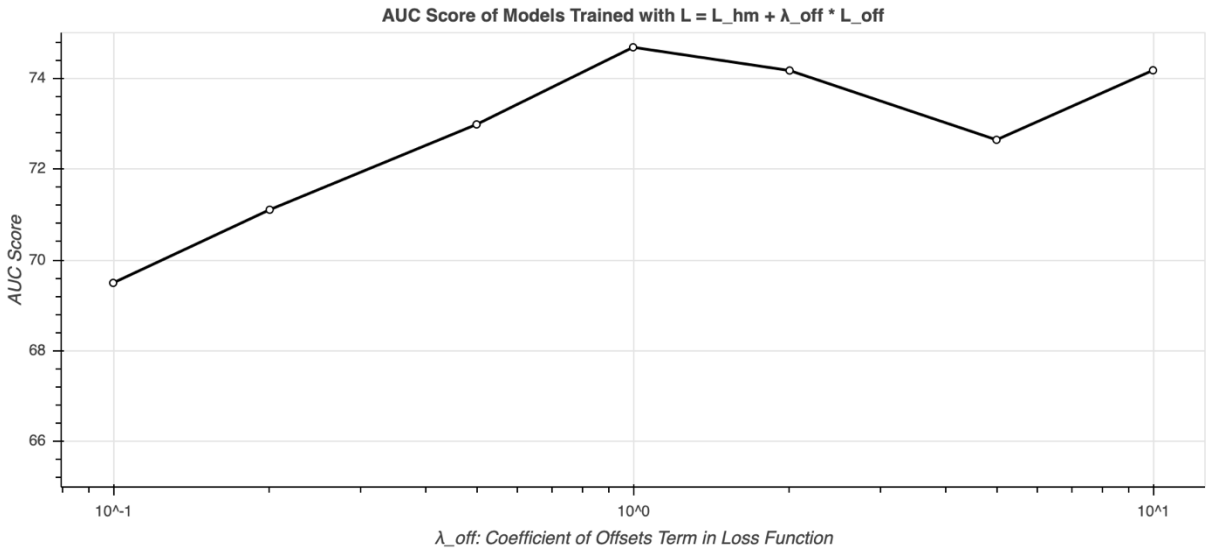

Figure 4.4.1 The tendency of AUC scores of models trained with different λ_{off}

The most effective value range of λ_{wh} is determined by disabling the offsets' loss ($\lambda_{off} = 0$) and train a series of models using the remaining loss function:

$$L = L_{hm} + \lambda_{wh} L_{wh},$$

where $\lambda_{wh} \in [0.1, 0.2, 1, 2, 10]$. Table 4.4.2 lists all the experiments and their results:

	λ_{wh}	λ_{off}	Precision	Recall	F1	AUC	AUC Score
Exp0009	10		0.181	0.244	0.208	0.15528167	6.2112668
Exp0010	5		0.183	0.219	0.199	0.14891	5.9564
Exp0011	1	0	0.176	0.284	0.217	0.1708168	6.832672
Exp0012	0.2		0.189	0.325	0.239	0.1946	7.784
Exp0013	0.1		0.123	0.229	0.16	0.127752	5.11008

Table 4.4.2 The performance of models trained with different λ_{wh}

From Table 4.4.2, we notice that there is no obvious pattern revealing how λ_{wh} affects the model's performance. Compared with other models at this stage, the model trained with $\lambda_{wh} = 0.2$ has a better result. Figure 4.4.2 compares two models trained with $\lambda_{off} = 1, 0$ and $\lambda_{wh} = 0, 0.2$. The latter's performance is frustrating and far below the average, which also

emphasizes the vital role of the prediction of the offsets.

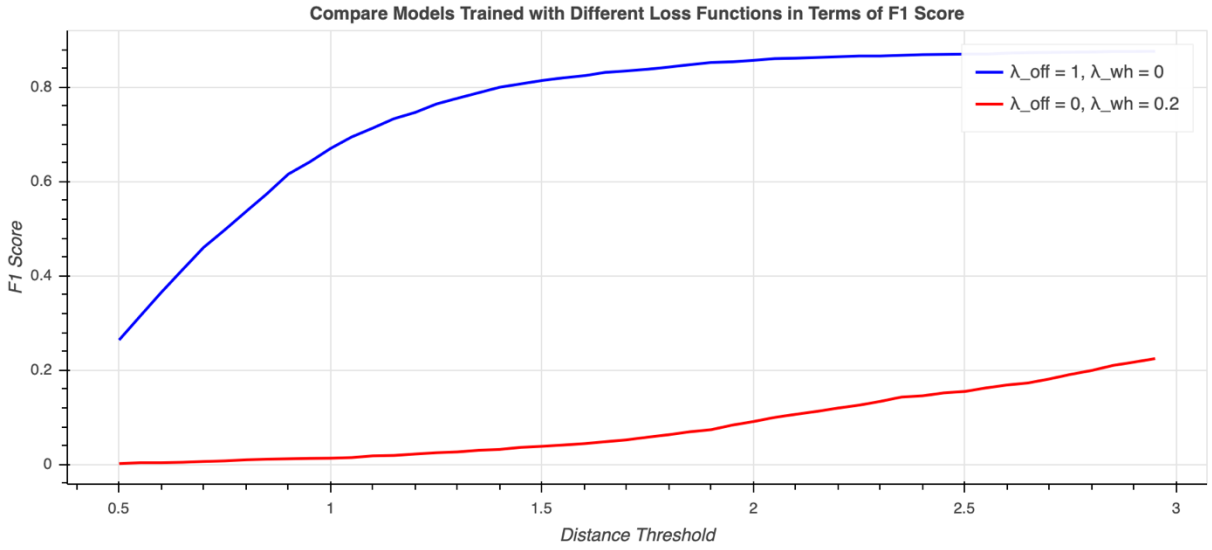

Figure 4.4.2 A comparison of the best-performing models in the first optimization phase

The previous experiments have shown that heatmap and offset losses play an equally essential role in tiny object detection. Therefore, we set $\lambda_{wh} = \lambda_{hm} = 1$. In order to find the global optimal solutions for λ_{off}, we need to conduct more experiments to refine their values. Thus, we train models with the complete loss function as:

$$L = L_{hm} + L_{wh} + \lambda_{off} L_{off},$$

where $\lambda_{off} \in [0.01, 0.1, 0.2, 0.5, 1]$. The performance of these models is recorded in Table 4.4.3. We also add the best-performing models from the previous two stages into the competition (See Figure 4.4.3). The model trained with loss function $L = L_{hm} + L_{wh} + 0.1 L_{off}$ outperforms the other models in all respects. Thus, this loss function is considered the most effective and will be used in the following explorations.

	λ_{wh}	λ_{off}	Precision	Recall	F1	AUC	AUC Score
Exp0014	1	1	0.871	0.817	0.843	1.8324706	73.298824
Exp0015	0.5	1	0.897	0.832	0.863	1.84716	73.8864
Exp0016	0.2	1	0.865	0.846	0.855	1.853711	74.14844

Exp0017	0.1	1	0.908	0.867	0.887	1.912329	76.49316
Exp0018	0.01	1	0.812	0.859	0.835	1.828738	73.14952

Table 4.4.3 The performance of models trained with different λ_{wh} and λ_{wh}

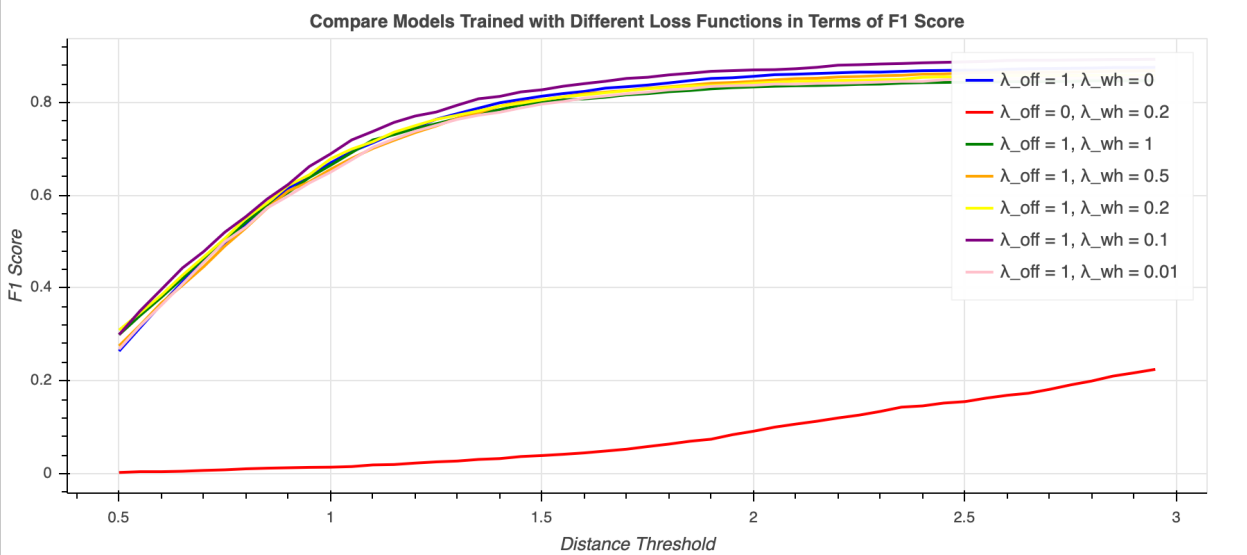

Figure 4.4.3 A Comparison of the best-performing models in the final optimizing phase

4.5 Resolution

Detecting efficiency is another critical perspective to compare the performance of models. Especially for practical applications like puck detection, we want to train a model that achieves great accuracy while keeping a reasonable speed. In order to evaluate the model's efficiency, we measure the average time that the model spends to process one frame. The inference duration contains the time used to extract representations for nine consecutive frames, refine feature maps with 3D CNN, and retrieve the final predictions. The data's quality has a significant impact on the model's inference time and accuracy. Generally, quality refers to the videos' resolution, the total number of pixels in each frame. As for the puck detection, giving a high-quality video means there is more information to leverage. Nevertheless, it also brings extra computations, which increase the inference time. By inputting higher resolution data, the

models can achieve higher accuracy at the cost of more extended training and inference time and vice versa.

In this series of experiments, we train and validate models using data with resolutions ranging from 128 x 128 to 896 x 896. Except for the resolution, we make sure all other configurations are constant. Table 4.5.1 shows the results of these experiments. The model's accuracy is represented by its AUC Score (F1 curves), and the model's efficiency is compared in terms of average inference duration.

	Resolution	AUC Score	Inference Duration (s)
Exp2001	896	83.21	3.696
Exp2002	768	76.07	2.980
Exp2003	640	78.08	2.841
Exp2004	512	76.49	2.729
Exp2005	384	68.08	2.730
Exp2006	256	50.76	2.656
Exp2007	128	4.52	2.668

Table 4.5.1 The effect of data's resolution on model's performance

The model's accuracy is positively associated with the quality of input data. We note that higher resolution gives the model more chance to precisely recognize and locate the tiny object in a video. However, the improvement of the AUC score becomes less significant when the resolution surpasses 512 x 512. Figure 4.5.1 shows the F1 score curves from all experiments. We can intuitively observe that the benefit from high-quality data reaches a bottleneck when the resolution is over 512. As we mentioned above, the model's high accuracy is usually obtained at the cost of efficiency. Since the model's performance in this perspective tends to stay steady after a certain threshold, it is not wise to brutally select the setting that gives the highest AUC score. We want to make a more informed and comprehensive decision, so we

involve comparisons in a new perspective: detecting efficiency.

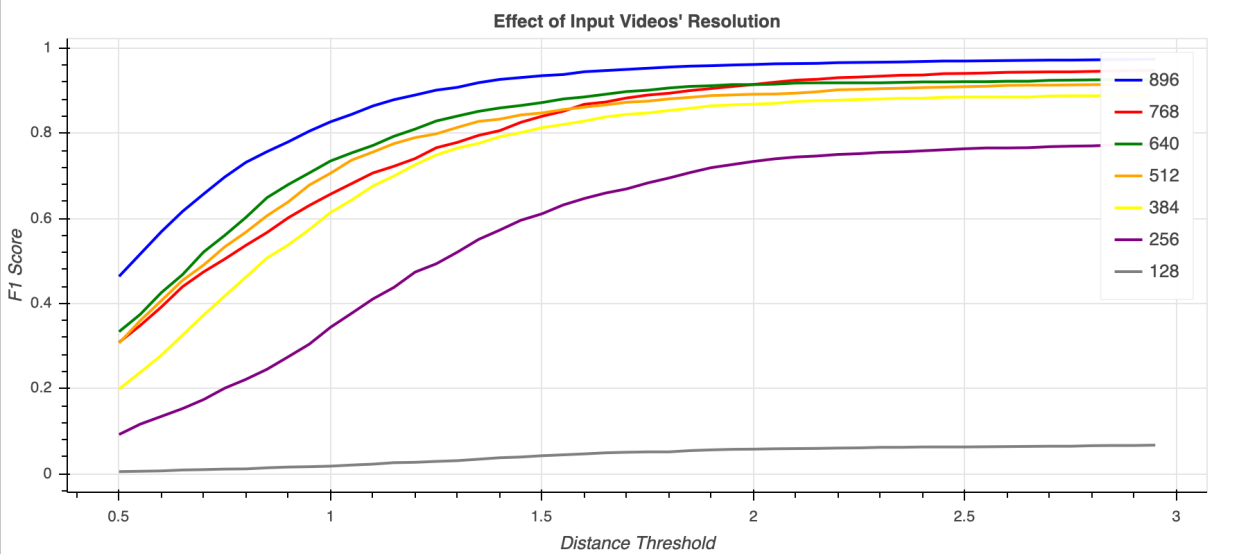

Figure 4.5.1 Comparing the accuracy of the models trained with different-quality data

We use 100 nine-frame video clips to measure each model's average inference time. Figure 4.5.2 explicitly depicts the difference between the models' efficiencies. Globally speaking, the results match our previous assumption that larger input data slower the inference. However, the influence of input data's quality on the model's efficiency fades out after a certain threshold. In this project, this threshold is resolution = 512 x 512.

The reason for the existence of such a bottleneck is that our model has a deep and wide architecture. No matter how tiny the input data are, there is a minimum operation time. Looking at the above factors, we determine that the resolution of both training and inference data should be 512.

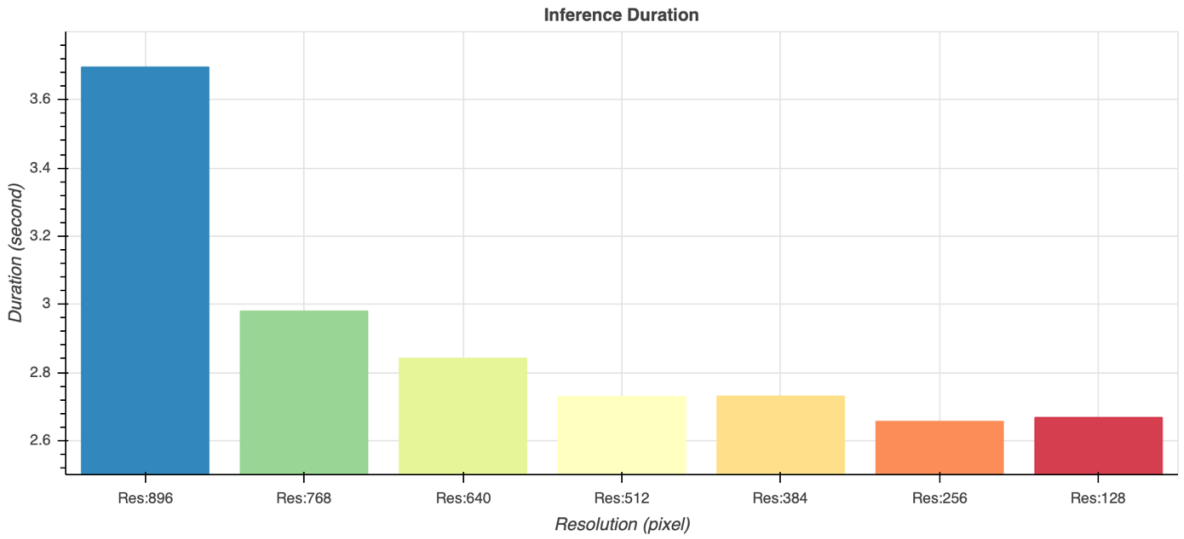

Figure 4.5.2 Comparing the detecting efficiency of models trained with different-quality data

In a word, considering this project's practical motivation, we want to pick a model that can have satisfactory performance in both perspectives: accuracy and efficiency. The quality of input data has a significant impact on them but in different ways. According to the results, high-resolution data enable the model to reach more accurate predictions but slower. Similarly, the model can process low-resolution input data faster at the cost of low accuracy. There is one thing in common: ceilings exist in both directions, which means the influence from the quality of data working on accuracy and efficiency is limited. To achieve either higher accuracy or faster detection, we need to focus on optimizing the model structure and training schemes. In the following research, we resize any video of any size to 512 x 512 pixels as a pre-processing method.

4.6 Summary

This chapter covers the procedures of selecting all of the important hyperparameters, model architectures, and training schemes for puck detection. By analyzing the loss and model's performance during training and validation, we note that the three subtasks have different

learning speeds. Globally speaking, the overfitting doesn't occur within 140 training epochs. Thus, the number of training iterations is set to 140. For the inference, the initial output includes a large number of predictions because of the unique model structure. We impose a series of post-processing filters to eliminate low-confidence and redundant predictions. The most effective thresholds are ScoreThre = 0.2 and NMSThre = 5. The model inherits the "backbone + multiple heads" structure, where the backbone has a huge impact on the model's performance. We compare three different backbones networks, and find Deep Layer Aggregation network outperforms others in a great way. Three heads are responsible for predicting the center point heatmap, object's shape, and offset. In order to train the model as an entirety, we implement the multi-task loss, the linear combination of three individual losses. Through a thorough exploration, we find the most productive loss function is $L = L_{hm} + L_{wh} + 0.1L_{off}$. Finally, we determine that input data with a resolution of 512 by 512 performs the best in all respects.

To summarize, we conducted a series of experiments to select the most effective and efficient hyperparameters, architectures, loss functions, and data quality. All of the filtered settings and configurations enable us to make the most use of the tiny object detection model. In Chapter 5, we display more examples of our model and compare it with other state-of-art object detectors in all respects in order to show its significant performance.

Chapter 5 Results and Discussion

In 5.1, we first compare our model with other state-of-art object detectors to show its strong performance in the field of tiny object detection. More examples of our model tackling challenging situations, such as motion blur, occlusion, and visual noise, are presented in 5.2. We objectively summarize the strength and weaknesses of our model and display its final performance in 5.3. The final conclusion and future research directions are discussed in 5.4.

5.1 Comparison With the State-of-the-Art Detectors

Our model aims to solve the problem of tiny and fast-moving object detection in videos, which fills a knowledge gap in the field of vision-based object detection. The state-of-art object detection models can be grouped into "one-stage" and "two-stage" in terms of the structure of the models they use. YOLOv3 [18] is one of the most successful one-stage object detection models that can achieve remarkable inference speed while maintaining relatively high accuracy. On the other hand, Mask RCNN [19], a two-stage detector, is famous for its significant detection accuracy. Both models are widely used in different applications. When it comes to tiny object detection, how do these models perform?

To explicitly show how much progress our model has made in the field of tiny object detection, we re-trained and evaluated the state-of-arts models using the same dataset (McGill Puck Dataset). The results are shown in Table 5.1.1. We note that our model outperforms YOLOv3 and Mask RCNN by a significant margin.

	Precision	Recall	F1 score
YOLOv3 [1]	0.534	0.954	0.685
Mask RCNN [2]	0.788	0.713	0.749
Ours	0.908	0.867	0.887

Table 5.1.1 Comparisons between our model and state-of-art object detectors

The Precision depicts how many predictions are correct. Our model achieves the

Precision of 0.9, which means more than 90 percent of predictions match the ground truth. However, the Precision of YOLOv3 and Mask RCNN only achieve 0.534 and 0.788, respectively. This implies that the state-of-art object detection models are likely to make more wrong predictions than ours for tiny objects. The reason for this gap is that our model wisely uses the temporal information embedded in the video sequence, while the others only leverage the information from a single image taken for a game video. We conclude from this that the temporal information can help to eliminate incorrect predictions.

The Recall indicates how much ground truth is precisely detected. Surprisingly, the Recall of YOLOv3 is higher than ours, meaning that YOLOv3 is less likely to miss the ground-truth puck in the frame. However, YOLOv3 produces a high Recall but a low precision because it tends to make a massive number of predictions per frame. Therefore, in order to compare the overall capability of the three models, we compute the F1 scores for each model. Our tiny object detection model achieves the F1 score of 0.887, while YOLOv3 and Mask RCNN reach 0.685 and 0.749. There is a more than a ten percent improvement using our model, which is a significant improvement in the state-of-the-art.

Moreover, Figure 5.1.1 – 5.1.4 show examples of these three object detection models dealing with a challenging case, in which a player's stick partially occludes the puck. In order to get a closer observation, we zoom-in the regions that include either a ground truth or a prediction (check the arrows and blue windows under the original frame). Figure 5.1.1 is the original frame, and the ground-truth puck is circled in a small green bounding box. We run our model, YOLOv3, and Mask RCNN on this example to explicitly compare these models' capabilities. As shown in Figure 5.1.2, our model makes a clean, confident, and correct prediction. Because of the analysis of temporal information, our model can precisely locate the occluded puck and avoid making incorrect predictions on the ice.

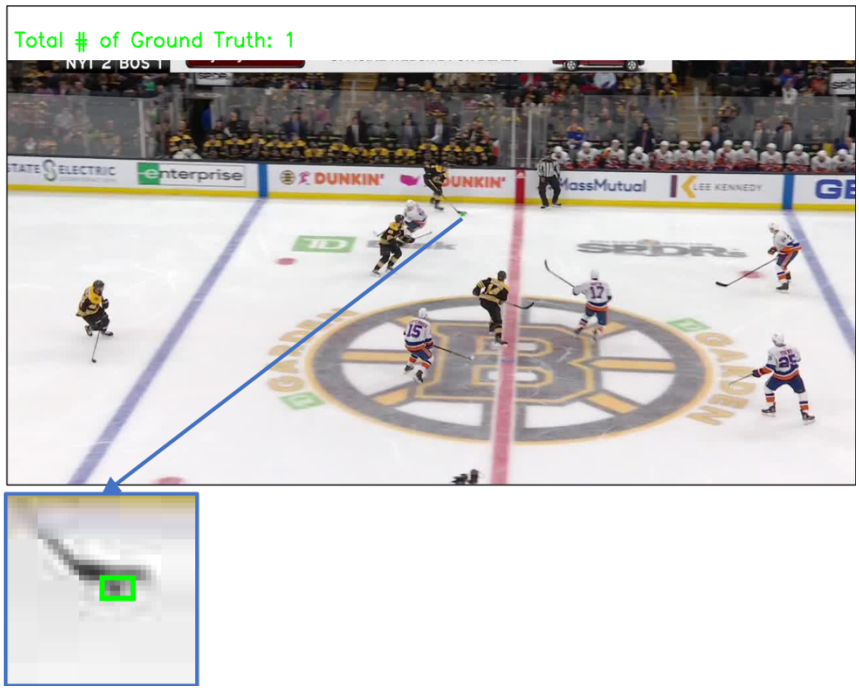

Figure 5.1.1 An example of an annotated frame. The green bounding box shows the ground-truth location of puck in this frame. In order to closely observe the results, we zoom in on the regions that contain either the predicted or ground-truth puck (shown in blue windows under the original frame).

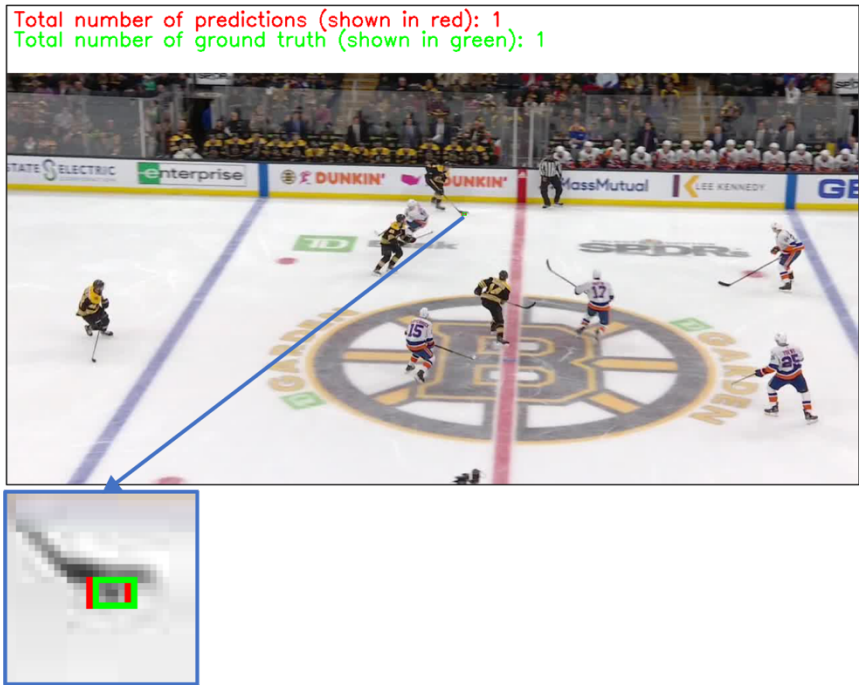

Figure 5.1.2 An example of the output of our model. The green and red bounding boxes show

the ground-truth and predicted locations of the puck in this frame. In order to closely observe the results, we zoom in on the regions that contain either the predicted or ground-truth puck (shown in blue windows under the original frame).

Figure 5.1.3 shows the output from Mask RCNN. The model successfully found the puck. Nevertheless, Mask RCNN made a false-positive prediction around the referee's skates (shown in the 2nd zoom-in window).

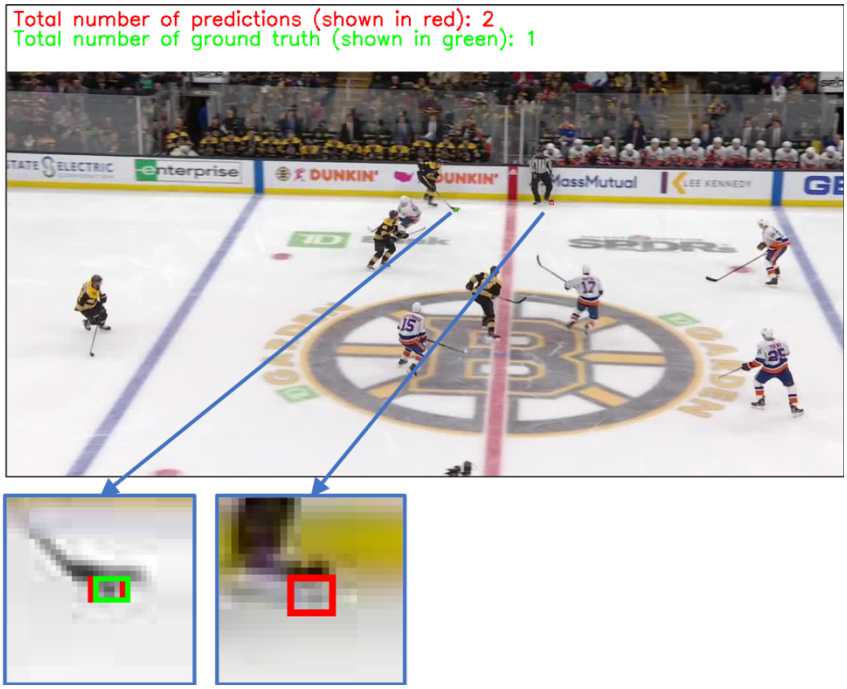

Figure 5.1.3 An example of the output of Mask RCNN. The green and red bounding boxes show the ground-truth and predicted locations of the puck in this frame. In order to closely observe the results, we zoom in on the regions that contain either the predicted or ground-truth puck (shown in blue windows under the original frame).

YOLOv3 can also locate the puck. The problem is that it makes too many other incorrect predictions. As shown in Figure 5.1.4, YOLOv3 wrongly identified the advertisements on the border (the 1st and 2nd windows), the skates of the referee (the 4th window), and a stick (the 5th window) as the puck. Thus, regardless all of them correctly find the location of the ground-truth puck, we conclude that the performance of our model working on “occlusion case” is better than the state-of-art detectors.

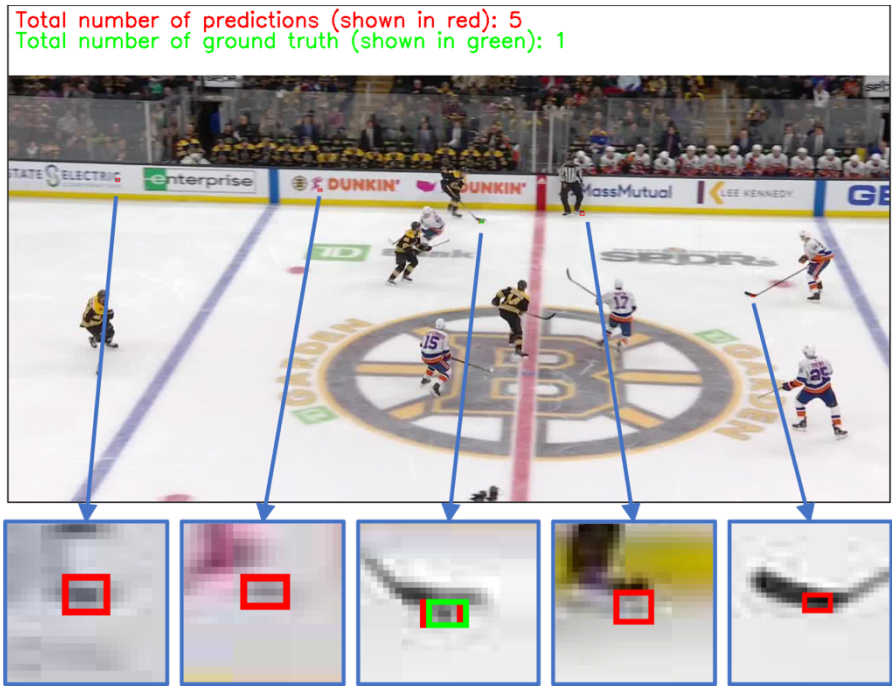

Figure 5.1.4 An example of the output of YOLOv3. The green and red bounding boxes show the ground-truth and predicted locations of the puck in this frame. In order to closely observe the results, we zoom in on the regions that contain either the predicted or ground-truth puck (shown in blue windows under the original frame).

To summarize, our model outperforms the state-of-art models in the field of tiny object detection. This new proposed model can maintain a relatively high Precision and Recall, so its overall performance is more remarkable.

5.2 Difficult Case Studies

The most difficult challenge for puck detection is the tiny size of the puck. Besides that, occlusion, motion blur, and visual noise are also challenges that we considered. In this section, we present examples of how these issues were dealt with in this thesis.

Consider the most common situation (shown in Figure 5.2.1), where there is neither occlusion nor motion blur. In this case, the difficulty is just the tiny size of the puck. In this situation, the puck maintains the standard puck's appearance. For frames containing such "clean" puck, all object detectors can successfully locate the puck.

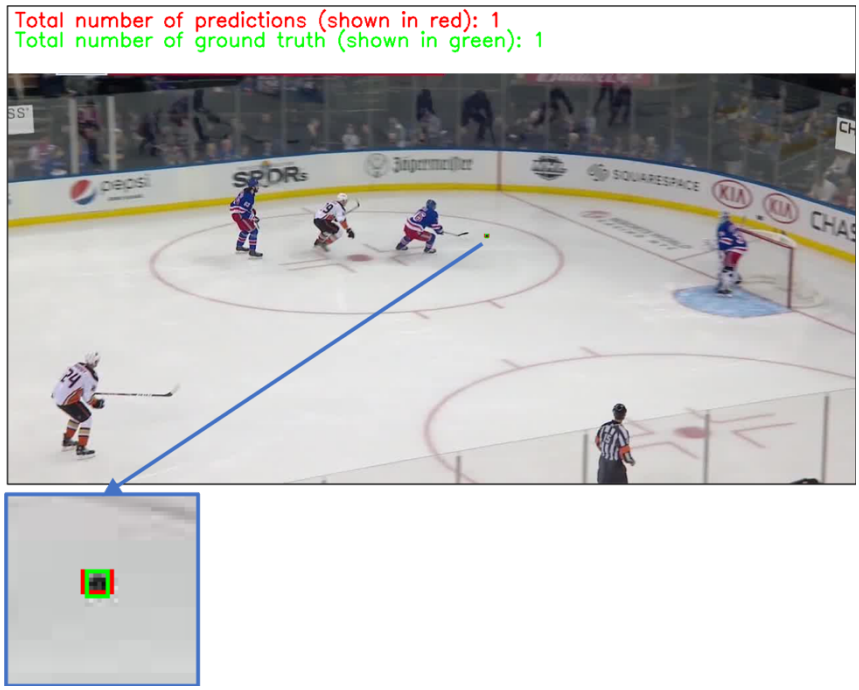

Figure 5.2.1 Case study: common situation. The green and red bounding boxes show the ground-truth and predicted locations of the puck in this frame. In order to closely observe the results, we zoom in on the regions that contain either the predicted or ground-truth puck (shown in blue windows under the original frame).

Figure 5.2.2 shows an example in which the puck is occluded by the player's stick. For this case, even a human can hardly find the puck at a glance. However, our model is able to precisely predict the location of the puck on the ice. This ability benefits from the use of the temporal information in videos. With the help of several frames before and after, our model can confidently make the correct prediction in this challenging case.

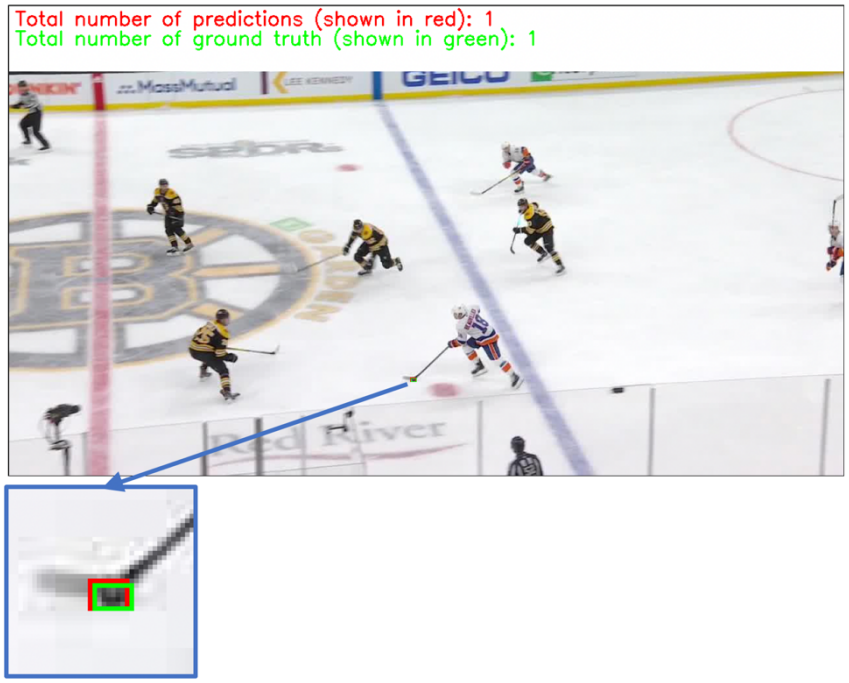

Figure 5.2.2 Case study: partial occlusion. The green and red bounding boxes show the ground-truth and predicted locations of the puck in this frame. In order to closely observe the results, we zoom in on the regions that contain either the predicted or ground-truth puck (shown in blue windows under the original frame).

The appearance of the puck is often hazy, usually referred to as motion blur. This case happens frequently in ice hockey broadcasts because the moving speed of the puck is dramatically high, and fast-moving objects appear as semi-transparent streaks. Thus, in order to improve the overall performance of puck detection, we need to make sure that our model is capable to precisely recognize the puck, regardless of motion blur. Figure 5.2.3 shows a case of motion blur occurring. After zooming in on the region including the puck, we note that visually detecting a blurred puck is extremely difficult. However, our model can precisely recognize this cluster of blurred pixels as the actual puck.

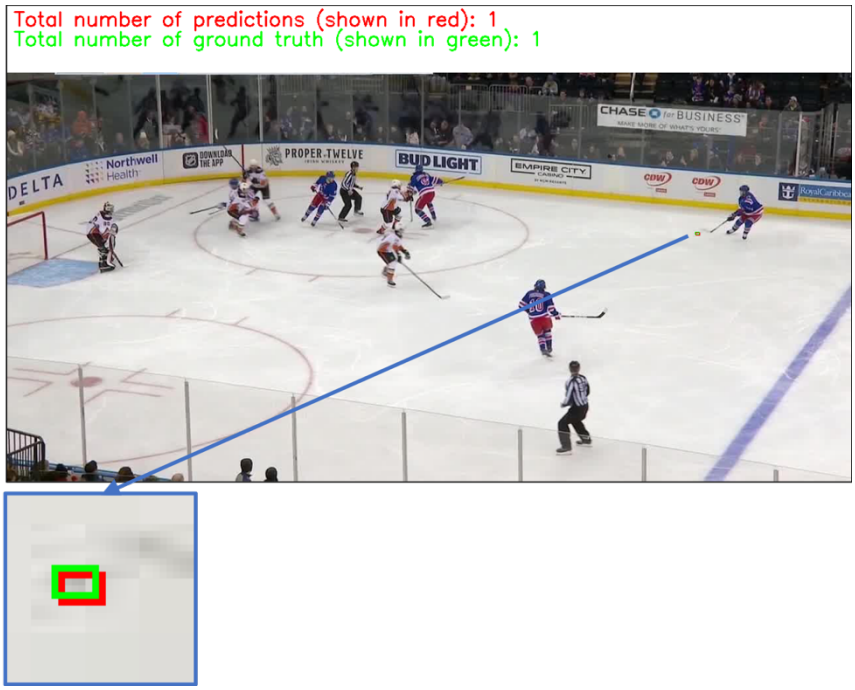

Figure 5.2.3 Case study: motion blur. The green and red bounding boxes show the ground-truth and predicted locations of the puck in this frame. In order to closely observe the results, we zoom in on the regions that contain either the predicted or ground-truth puck (shown in blue windows under the original frame).

Another issue related to puck detection is that there are many lines, circles, and even advertisements painted on the ice. All of these, referred to as visual noise, can be distracting for puck detection when the puck passes over them. For example, as shown in Figure 5.2.4, when the puck is precisely on the line, it is impossible to find it merely only obtaining information from one frame. Our model deduces a possible location of the puck in this frame by analyzing four frames before and after. It turns out that this inference successfully matches the ground truth.

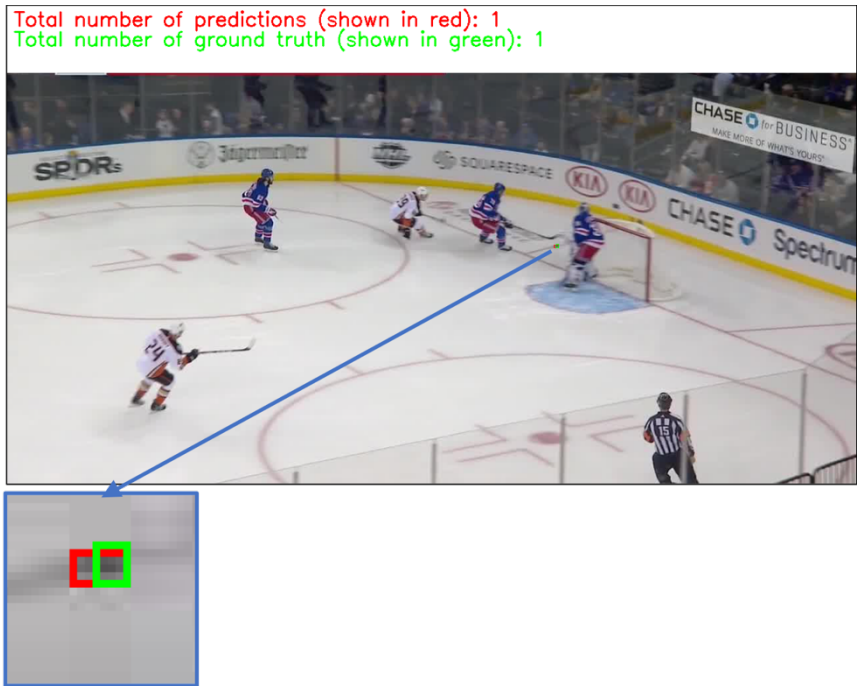

Figure 5.2.4 Case study: visual noise. The green and red bounding boxes show the ground-truth and predicted locations of the puck in this frame. In order to closely observe the results, we zoom in on the regions that contain either the predicted or ground-truth puck (shown in blue windows under the original frame).

5.3 Strengths and Weaknesses

Our model achieves the best extant performance in the field of tiny object detection. It outperforms all of the state-of-art object detectors by a large margin. By leveraging both semantic information from still images and temporal information from videos, this novel tiny object detector can effectively tackle such challenges as occlusion, motion blur, visual noise, and tiny size. Table 5.3.1 presents the final performance of our model when evaluated on the validation and test sets of the McGill Puck Dataset. The performance on these two sets is very similar, so we conclude that there is no evidence of overfitting.

	Precision	Recall	F1 score	AUC score
Validation	0.908	0.867	0.887	76.493
Test	0.911	0.873	0.892	76.527

Table 5.3.1 The model's performance on the validation and test sets

This model focuses on using the temporal information obtained from broadcast hockey videos in addition to the summarized features from individual (static) images frames to perform tiny object detection. Converting traditional object detection to center point detection enables the model to focus on finding the location of a tiny object instead of predicting its appearance. Also, the particular evaluation method can be applied to other tiny object detection tasks.

The drawbacks of this model are also apparent. Since this model implements a multi-stage structure, the training and inference cannot be completed end-to-end. We need to train each component individually. This property increases the cost of training and inference in terms of both time and computational resources. The average inference time for one frame is 2.729 seconds, which includes the time for Stage One that computes the representations of nine frames (2.412 seconds) and Stage Two that computes the temporal information (0.317 seconds). Currently, this model cannot support real-time detection because it requires information from the "future" frames. For example, in order to detect the puck in the frame t = T, the input data must include frames from t = T -4 to t = T + 4. Thus, there is always at least four-frame delay for this model. We discuss about the potential solutions to optimize the inference processing in the following section. Additionally, in order to connect Stage One and Stage Two, the model must save and reload the summarized representations of all frames, which is extremely time-consuming but inevitable. Therefore, the model cannot achieve a near real-time detection. In fact, there will be an approximately 2.7 seconds delay for each frame.

5.4 Conclusion and Discussion

In this work, we presented a deep-learning approach to locate the hockey puck in NHL broadcast videos. The core model consists of two sequential stages. In the first stage, an upgraded Deep Layer Aggregation network is used to extract the summarized representation of each frame. In the second stage, a three-layer 3D convolutional neural network is implemented to compute the temporal information embedded in videos. To increase the accuracy, we abandoned the traditional "region proposal + bounding box regression" mode [13] and

converted the problem to a task of “center point detection”. One particular convolutional block calculates the possibility of each pixel being the center point of an object. Another two convolutional blocks predict the corresponding shape and offset. The whole model implements a classical “backbone + multiple heads” structure [34], where Deep Layer Aggregation and 3D convolutional neural networks are the backbones, and three convolutional blocks are the heads. The final prediction is a combination of the outputs of the three heads. After a series of thorough explorations, we selected the most effective and efficient hyper-parameters, backbone structure, input resolution, training scheme for our tiny and fast-moving object detection model.

We also collected McGill Puck Dataset that contains 130 video clips and 17,997 frames from NHL broadcast videos. The puck is the only annotated object in the dataset. MPD enriches the hierarchy of object detection in terms of the object’s size. Additionally, other researchers can examine and compare their tiny object detection algorithms using MPD, which benefits applications requiring tiny object detection. We also updated the criteria used to determine the correctness of the predictions. Instead of computing the Intersection over Union between bounding boxes, we compare the L2 distance between the objects’ center points.

Overall, our model achieves the Precision of 0.908 and the Recall of 0.867. The F1 score reaches 0.887, which is significantly higher than the performance of YOLOv3 [18] (F1 score = 0.685) and Mask RCNN [19] (F1 score = 0.749) on the McGill Puck Dataset. This impressive performance indicates that tiny object detection requires a different model from other larger size object detectors. With careful refinement and optimization, our model outperforms the state-of-art models in the field of tiny object detection. The development of this branch of object detection may boost applications requiring to locate and track tiny objects in videos, such as missile tracking, border surveillance, etc. However, the inference speed of our model is not satisfactory. There is still much work to be performed to achieve real-time tiny object detection. There is also a concern about how to incorporate the algorithm in other practical applications.

Finally, there are additional research directions that are pertinent to this research:

1. Improve the performance of the individual components. There are several

unique components in our model. For example, the second feature engine is a 3D CNN, focusing on extracting temporal information from videos. It requires the summarized representations of all frames. Thus, the representations of some frames are repeatedly computed and stored. In order to predict the puck in the frame t = T, we need to compute the representations of all the frames in the range of [T – 4, T + 4]. Since the resolution of these summarized representations is 64 x 128 x 128, we need a large amount of space to store these intermediate products. The complete saving and reloading procedures require a large amount of time. This mechanism brings redundant computations, severely slowing the whole system. There are other options like bi-directional LSTM [43] that can better process sequential data. With the development of other algorithms [44 - 46], we can replace the existing constituents with more efficient models to further boost overall performance. Also, accelerating the inference time is quite essential for all practical applications. The current model cannot achieve real-time detection due to its complicated structure and mechanisms. However, the model cannot obtain such satisfactory performance on tiny object detection without any of the existing components. Currently, no solution enables the model to achieve both high accuracy and fast detection. The ultimate goal is to design a model achieving real-time tiny detection with high accuracy.

2. Involve semi-supervised learning. Just like most state-of-art object detectors [18, 19], ours is a supervised learning model. The number of labelled data has a significant impact on the model's final performance. In contrast, a semi-supervised learning algorithm would demand only a small number of labelled data. If we wish to use our original model in other applications, we need to collect a new dataset and re-train the model from the beginning. However, human annotation is a hugely time-consuming task. An alternative is to label a small part of the dataset for a semi-supervised learning model [46]. This would greatly

accelerate the process of implementing state-of-art detection algorithms in other custom applications.

References

[1] Lin TY, Maire M, Belongie S, Hays J, Perona P, Ramanan D, Dollár P, Zitnick CL. Microsoft coco: Common objects in context. InEuropean conference on computer vision 2014 Sep 6 (pp. 740-755). Springer, Cham.

[2] Redmon J, Farhadi A. Yolov3: An incremental improvement. arXiv preprint arXiv:1804.02767. 2018 Apr 8.

[3] He K, Gkioxari G, Dollár P, Girshick R. Mask r-cnn. InProceedings of the IEEE international conference on computer vision 2017 (pp. 2961-2969).

[4] Wikipedia contributor. Ice hockey [Internet]. Wikipedia. Wikimedia Foundation; 2001. Available from: https://en.wikipedia.org/wiki/Ice_hockey

[5] Ren S, He K, Girshick R, Sun J. Faster r-cnn: Towards real-time object detection with region proposal networks. IEEE transactions on pattern analysis and machine intelligence. 2016 Jun 6;39(6):1137-49.

[6] Long J, Shelhamer E, Darrell T. Fully convolutional networks for semantic segmentation. InProceedings of the IEEE conference on computer vision and pattern recognition 2015 (pp. 3431-3440).

[7] Redmon J, Divvala S, Girshick R, Farhadi A. You only look once: Unified, real-time object detection. InProceedings of the IEEE conference on computer vision and pattern recognition 2016 (pp. 779-788).

[8] Everingham M, Van Gool L, Williams CK, Winn J, Zisserman A. The pascal visual object classes (voc) challenge. International journal of computer vision. 2010 Jun;88(2):303-38.

[9] Tono A. What is a multi-headed model? And what exactly is a 'head' in a model? [Internet]. Stack Overflow; 2012. Available from: https://stackoverflow.com/questions/56004483/what-is-a-multi-headed-model-and-what-exactly-is-a-head-in-a-model

[10] Vats K, McNally W, Dulhanty C, Lin ZQ, Clausi DA, Zelek J. PuckNet: Estimating hockey puck location from broadcast video. arXiv preprint arXiv:1912.05107. 2019 Dec 11.

[11] Blicher A. The history of analytics and statistics in tennis [Internet]. Medium. 2018. Available from: https://medium.com/the-sports-scientist/the-history-of-analytics-and-statistics-in-tennis-e19aa206fdf0

[12] Bogert T. Soccer Analytics: Three real-life examples that show how analysts impact their clubs [Internet]. MLSsoccer.com. 2015.Available from: https://www.mlssoccer.com/soccer-analytics-guide/2020/three-real-life-examples

[13] Carling C. MCFC Analytics - The Video Analyst.com [Internet]. The Video Analyst.com. 2011. Available from: https://thevideoanalyst.com/mcfc-analytics/

[14] Statsports. World Leading GPS Tracker & Sports Performance Analysis [Internet]. Statsports.com. 2020. Available from: https://statsports.com

[15] SportVU [Internet]. En.wikipedia.org. 2016.Available from: https://en.wikipedia.org/wiki/SportVU

[16] Kristan M, Matas J, Leonardis A, Felsberg M, Pflugfelder R, Kamarainen JK, Cehovin Zajc L, Drbohlav O, Lukezic A, Berg A, Eldesokey A. The seventh visual object tracking vot2019 challenge

results. InProceedings of the IEEE International Conference on Computer Vision Workshops 2019 (pp. 0-0).

[17] Mitzel D, Leibe B. Real-time multi-person tracking with detector assisted structure propagation. In2011 IEEE International Conference on Computer Vision Workshops (ICCV Workshops) 2011 Nov 6 (pp. 974-981). IEEE.

[18] Breitenstein MD, Reichlin F, Leibe B, Koller-Meier E, Van Gool L. Robust tracking-by-detection using a detector confidence particle filter. In2009 IEEE 12th International Conference on Computer Vision 2009 Sep 29 (pp. 1515-1522). IEEE.

[19] Wu D, Zheng SJ, Zhang XP, Yuan CA, Cheng F, Zhao Y, Lin YJ, Zhao ZQ, Jiang YL, Huang DS. Deep learning-based methods for person re-identification: A comprehensive review. Neurocomputing. 2019 Apr 14;337:354-71.

[20] Wu S, Chen YC, Li X, Wu AC, You JJ, Zheng WS. An enhanced deep feature representation for person re-identification. In2016 IEEE winter conference on applications of computer vision (WACV) 2016 Mar 7 (pp. 1-8). IEEE.

[21] Fani M, Neher H, Clausi DA, Wong A, Zelek J. Hockey action recognition via integrated stacked hourglass network. InProceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops 2017 (pp. 29-37).

[22] Lu WL, Okuma K, Little JJ. Tracking and recognizing actions of multiple hockey players using the boosted particle filter. Image and Vision Computing. 2009 Jan 1;27(1-2):189-205.

[23] Cavallaro R. The FoxTrax hockey puck tracking system. IEEE Computer Graphics and Applications. 1997 Mar;17(2):6-12.

[24] Contributors R. One-Stage Detectors and Two-Stage Detectors [Internet]. Reddit.com. 2017. Available from: https://www.reddit.com/r/MachineLearning/comments/e9nm6b/d_what_is_the_definition_of_onestage_vs_twostage/

[25] Girshick R, Donahue J, Darrell T, Malik J. Region-based convolutional networks for accurate object detection and segmentation. IEEE transactions on pattern analysis and machine intelligence. 2015 May 25;38(1):142-58.

[26] Uijlings JR, Van De Sande KE, Gevers T, Smeulders AW. Selective search for object recognition. International journal of computer vision. 2013 Sep 1;104(2):154-71.

[27] He K, Zhang X, Ren S, Sun J. Spatial pyramid pooling in deep convolutional networks for visual recognition. IEEE transactions on pattern analysis and machine intelligence. 2015 Jan 9;37(9):1904-16.

[28] Girshick R. Fast r-cnn. InProceedings of the IEEE international conference on computer vision 2015 (pp. 1440-1448).

[29] Lin TY, Dollár P, Girshick R, He K, Hariharan B, Belongie S. Feature pyramid networks for object detection. InProceedings of the IEEE conference on computer vision and pattern recognition 2017 (pp. 2117-2125).

[30] Han W, Khorrami P, Paine TL, Ramachandran P, Babaeizadeh M, Shi H, Li J, Yan S, Huang TS. Seq-nms for video object detection. arXiv preprint arXiv:1602.08465. 2016 Feb 26.

[31] Broad A, Jones M, Lee TY. Recurrent Multi-frame Single Shot Detector for Video Object Detection. InBMVC 2018 (p. 94).

[32] Zhu X, Wang Y, Dai J, Yuan L, Wei Y. Flow-guided feature aggregation for video object detection. InProceedings of the IEEE International Conference on Computer Vision 2017 (pp. 408-417).

[33] Open CV. openvinotoolkit/cvat [Internet]. GitHub. 2016. Available from: https://github.com/openvinotoolkit/cvat

[34] Yu F, Wang D, Shelhamer E, Darrell T. Deep layer aggregation. InProceedings of the IEEE conference on computer vision and pattern recognition 2018 (pp. 2403-2412).

[35] LeCun Y. The MNIST database of handwritten digits. http://yann. lecun. com/exdb/mnist/. 1998.

[36] Zhang X, Zou J, He K, Sun J. Accelerating very deep convolutional networks for classification and detection. IEEE transactions on pattern analysis and machine intelligence. 2015 Nov 20;38(10):1943-55.

[37] He K, Zhang X, Ren S, Sun J. Deep residual learning for image recognition. InProceedings of the IEEE conference on computer vision and pattern recognition 2016 (pp. 770-778).

[38] Donahue J, Jia Y, Vinyals O, Hoffman J, Zhang N, Tzeng E, Darrell T. A deep convolutional activation feature for generic visual recognition. UC Berkeley & ICSI, Berkeley, CA, USA.

[39] Long J, Shelhamer E, Darrell T. Fully convolutional networks for semantic segmentation. InProceedings of the IEEE conference on computer vision and pattern recognition 2015 (pp. 3431-3440).

[40] Ioffe S, Szegedy C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. arXiv preprint arXiv:1502.03167. 2015 Feb 11.

[41] Agarap AF. Deep learning using rectified linear units (relu). arXiv preprint arXiv:1803.08375. 2018 Mar 22.

[42] Law H, Deng J. Cornernet: Detecting objects as paired keypoints. InProceedings of the European Conference on Computer Vision (ECCV) 2018 (pp. 734-750).

[43] Developer N. NVIDIA TITAN RTX is Here [Internet]. NVIDIA. 2020. Available from: https://www.nvidia.com/en-us/deep-learning-ai/products/titan-rtx/

[44] Huang Z, Xu W, Yu K. Bidirectional LSTM-CRF models for sequence tagging. arXiv preprint arXiv:1508.01991. 2015 Aug 9.

[45] Jeong J, Lee S, Kim J, Kwak N. Consistency-based semi-supervised learning for object detection. InAdvances in neural information processing systems 2019 (pp. 10759-10768).