Hochschule
Bonn-Rhein-Sieg
University of Applied Sciences

b-it Bonn-Aachen International Center for Information Technology

R&D Project

Multi-Input Conditional Object Detection

Gokul Krishna Gandhi Chenchani

Submitted to Hochschule Bonn-Rhein-Sieg,
Department of Computer Science
in partial fullfilment of the requirements for the degree
of Master of Science in Autonomous Systems

Supervised by

Prof. Dr.-Ing. Sebastian Houben
M.Sc. Deebul Sivarajan Nair

August 2024

I, the undersigned below, declare that this work has not previously been submitted to this or any other university and that it is, unless otherwise stated, entirely my own work. The report was, in part, written with the help of the AI assistant ChatGPT-4, QuillBot and Perplexity as described in the appendix. I am aware that content generated by AI systems is no substitute for careful scientific work, which is why all AI-generated content has been critically reviewed by me, and I take full responsibility for it..

Date	Gokul Krishna Gandhi Chenchani

Abstract

This research introduces a novel Multi-Input Conditional Object Detection framework aimed at improving object detection by focusing on specific target objects, guided by conditional inputs such as support images or encoded labels. Unlike traditional object detection methods that process all objects within a scene, this approach is designed to be particularly effective in scenarios where only one object requires attention, such as in industrial applications like the RoboCup@Work competition. By narrowing the focus to a single object, this method addresses the inefficiencies and inaccuracies often encountered in complex environments with multiple objects.

The methodology behind this research involved creating a specialized dataset for the RoboCup@Work competition, featuring 18 distinct object classes. This dataset was meticulously annotated and processed using advanced image processing techniques to ensure that the training data was of the highest quality. The research further explores the integration of deep learning architectures, including ResNet50, ResNet18, and SSD, with different data fusion techniques like concatenation, addition, and multiplication. These fusion methods effectively combine image features with encoded labels, enabling the detection framework to operate more efficiently and accurately.

The results of the study demonstrate the conditional object detection framework in computational efficiency and accuracy. This research not only advances the current state of object detection but also opens new possibilities for its application across various fields that demand precise and efficient visual recognition. Future work could involve refining the model further, exploring additional fusion techniques, and applying the framework to a broader range of real-world scenarios, enhancing its adaptability and impact.

Acknowledgements

I would like to sincerely thank my supervisors Prof. Dr.-Ing. Sebastian Houben and M.Sc. Deebul Sivarajan Nair for their constant support and guidance during this project. Their contributions and insights were very helpful in shaping my project. I am honored to have had the opportunity to work with them.

I would also like to thank Nandhini Shree Mathivanan for her contributions for keypoint regression loss (Laplace loss) which was used in this project as one of the functionality.

I would like to thank the constant support of my friends from MAS and others who helped me to focus and gave inputs whenever needed. I would also like to thank my family for their unwavering moral support throughout the phase of this project.

Contents

List of Figures

List of Tables

1

Introduction

Object detection is one of the important functionality in computer vision and deep learning for a system to understand and analyze the visual environment [1]. Object detection in specific involves the identification and location coordinates of an object within a given input, as shown in figure 1.1. The significance of object detection lies in its wide array of applications, from autonomous driving and surveillance to robotics and medical imaging. This project focuses on the methodologies and advancements in object detection with a focus on the emerging approach of conditional object detection.

Methodologies in Object Detection:

Object detection methods have changed a lot over the years as novel methods have been found to make them more accurate and useful. Most of the time, bounding box regression, single key-point recognition, and center key-point identification with pose orientation are used.

- **Bounding Box Regression:** This technique involves generating four sets of measurements that form a rectangle around an item in a two-dimensional image. Usually, the numbers show where the top-left and bottom-right corners of the object's containing box are. This method is often used because it is easy to understand and works well in a number of situations, such as face detection and pedestrian detection [2].

- **Single Key-Point Detection:** Single key-point detection, in contrast to bounding box regression, concentrates on detecting the object's center. This method works well in situations when an object's precise location is more important than its surroundings. It is often used in situations when the object's center point offers enough details for further analysis, such tracking and action detection [3].

- **Center Key-Point Detection with Pose Orientation:** This advanced technique first identifies the object's central point and then uses that knowledge to generate a 3D bounding box and the object's pose in six degrees of freedom (6-DoF). This method is particularly effective in applications that require specific spatial information, such as robot handling and augmented reality [4].

Despite improvements in accuracy and efficacy, these techniques pose obstacles, particularly when working with images containing many objects from several classes. A trained model typically creates a big output vector that includes all recognized items in the input image. However, in real-world applications such as robotic manipulation, it is typically necessary to manipulate only one target object at a time.

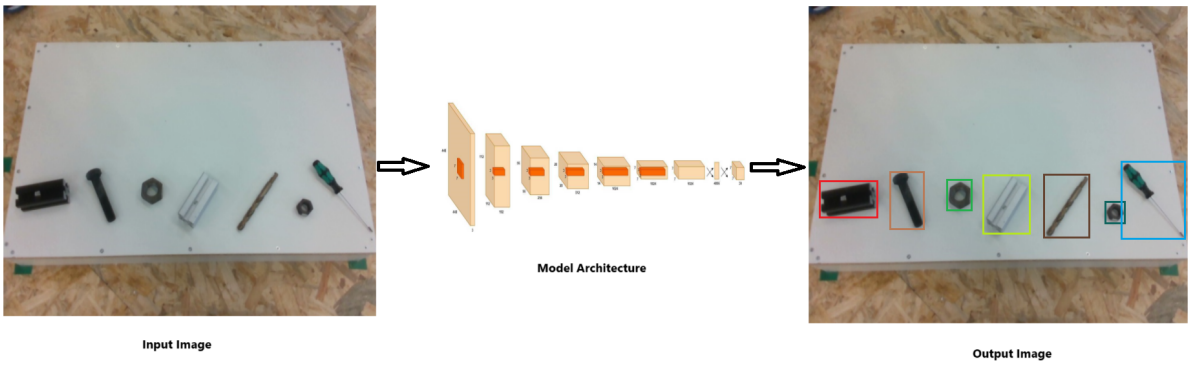

Figure 1.1: General pipeline of traditional object detection

For example, in the RoboCup@Work competition [5], a mobile robot with a manipulator must move particular objects from one workstation to another. The task planner specifies that the manipulator can only handle one object per action. Detecting every object in such instances increases computing complexity unnecessarily, as the additional data adds nothing to the current job and instead burdens the system.

One possible solution for this problem seems to be conditional object detection. Conditional object detection searches for objects similar to a given support image unlike classical object detection, which recognizes every object within pre-trained categories. This method's diversity and use will grow as it detects objects from categories it has not before come across. Trained on pairs of support and query images, conditional object detection allows the model learn to categorize objects depending on certain criteria instead of a predefined list of categories. This training approach enhances the generalization capacity and adaptability of the model toward new objects. While normal object detection is examined using a lot of testing images [6], conditional object detection is tested using various pairs of support and query images to determine its success in detecting conditioned objects.

In this R&D project, we propose an approach to use conditional object detection to improve the efficiency and effectiveness of object detection in scenarios requiring single object manipulation, where an input condition such as an encoded label is passed along with the image to the model and expected output is the detection of the object based on the input condition as shown in figure 1.2. A model block diagram of both object detection and conditional object detection is shown in the figures 1.3.

1.1 Motivation

Many fields, including autonomous systems, surveillance, and healthcare, rely critically on object detection [7] [8]. Especially in applications like robotic manipulation when just a single target object requires attention, traditional object detection techniques can identify several objects in a scene, which can be computationally demanding and insignificant [9]. Robots assigned missions to move particular objects between workstations, for example, in the RoboCup@Work competition [5] finding additional objects affects task performance in addition to adding computational overhead.

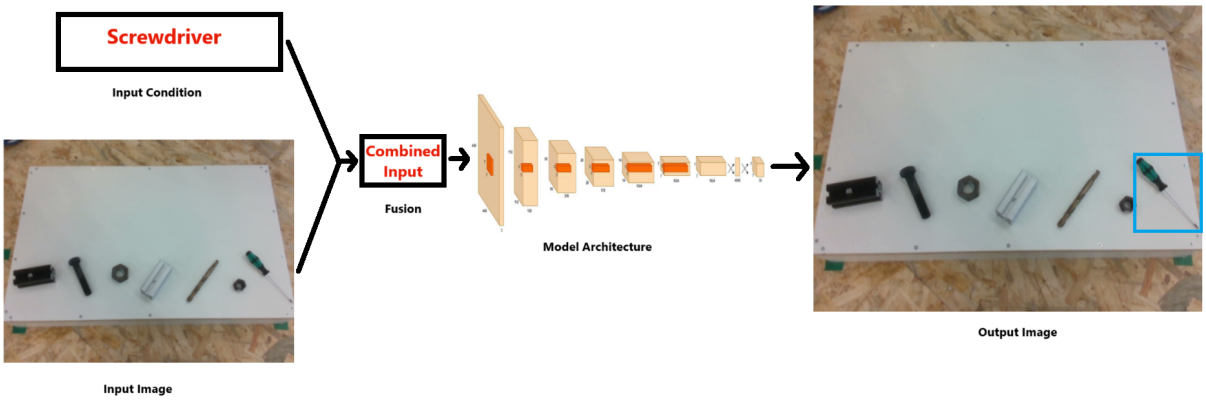

Figure 1.2: General pipeline of proposed conditional object detection

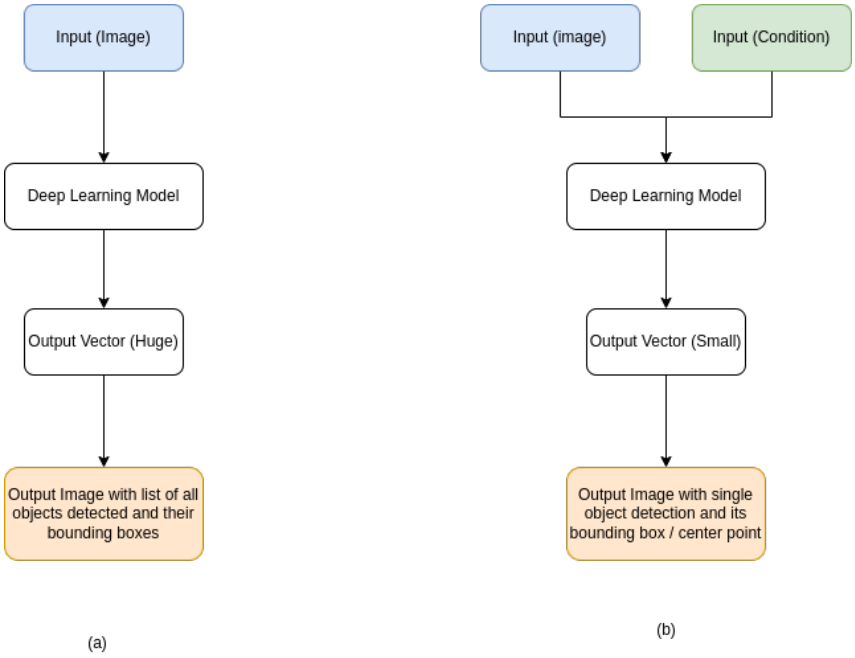

Figure 1.3: (a) Block diagram representation of object detection. (b)Block diagram representation of conditional object detection.

By concentrating only on locating a target object depending on a specific condition or support image, conditional object detection solves this inefficacy [10]. This method guarantees faster, more accurate detections by precisely matching tasks requiring manipulation of one object at a time with computing efficiency. Moreover, unlike conventional approaches limited to pre-trained categories, conditional object detection can adapt to new object categories by learning from support-query image pairs, so being highly adaptable in dynamic environments like industrial automation where object types often change [11].

Focusing on a single target also improves recognition accuracy by cutting down on false positives, which is especially helpful in scenes with a lot of things going on or where things are blocked out. Using a support image makes it easier for the model to distinguish the target object apart from other objects and the background [12].

In summary, the motivation for this research is to address the shortcomings and inadequacies of traditional object detection techniques in particular real-world contexts. Our research on conditional object detection seeks to improve object detection's computational effectiveness, flexibility, precision, and resilience. This will ultimately help progress autonomous systems, robotics, and other domains that depend on accurate and effective object detection.

1.2 Challenges and Difficulties

Traditional object detection still faces several challenges, even with developments. Among the various elements causing these problems are the complexity of visual environments, limited computational capability, and high demand for fast processing and great accuracy. Understanding these challenges can help one to design more dependable and effective object identification models.

1.2.1 Scale Variation

Objects can show different scales within images, which makes consistent detection challenging. Models have to be able to identify both small and large objects, a challenge for which conventional techniques usually fail [13].

1.2.2 Occlusion

Correct detection is complicated by partial or complete blocking of objects by other elements in a scene. Models must deduce the existence of blocked items, hence adding complexity [14].

1.2.3 Cluttered Backgrounds

Heterogeneous backgrounds can cause uncertainty in detection systems, thereby producing inaccuracies. In real-world settings, especially distinguishing target objects from meaningless background parts is quite difficult [15].

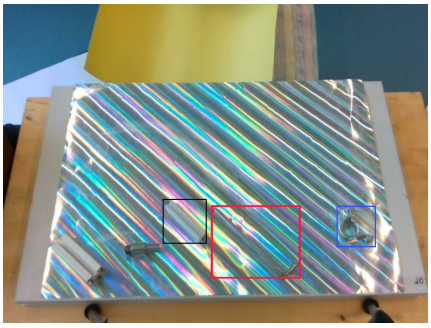

Figure 1.4: Object detection on arbitrary surface 1

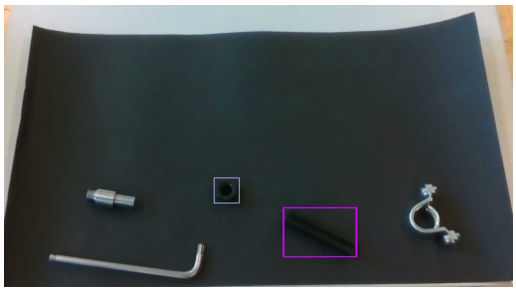

Figure 1.5: Object detection on arbitrary surface 2

1.2.4 Real-Time Processing

Robots and autonomous driving among other things need real-time object detection. Given the computing requirements of deep learning models [16], reaching this while preserving great accuracy is difficult.

1.3 Problem Statement

Traditional object detection techniques, including those used in current robotic systems, have major difficulties in settings when objects are put on random surfaces. These techniques specifically find difficulty with inaccurate object detection on such surfaces, which results in misclassification or inability to detect objects entirely [7] [16]. When the surface includes intricate patterns, reflections, or uneven textures—as seen in the examples where objects are incorrectly identified on patterned and reflecting surfaces—this problem gets even worse.

Furthermore, the current approach performs ineffective tensor computation. Even if the task just requires one object, traditional models process all possible object classes in an image [8]. This causes unnecessary computational overhead since every recognized object produces an output tensor comprising several bounding boxes and class predictions, hence increasing the processing time and complexity without adding value to the particular task at hand.

This work aims to investigate conditional object detection as a solution for these challenges. Focusing the detection process on a single target object designated by a particular condition or label can help the model to possibly:

1. **Improve Accuracy:**
    - Conditioning the detection on specific objects helps the model handle more complicated and arbitrary surfaces, hence lowering misclassification and increasing detection accuracy [10] [11] as shown in figures 1.4 1.5.

2. **Enhance Computational Efficiency:**

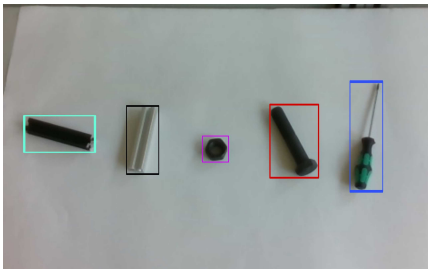

Figure 1.6: Detection of all objects

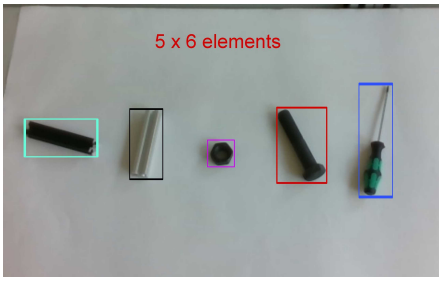

Figure 1.7: Output tensor of performed object detection

- Conditional detection lets the model overlook pointless objects, hence lowering the output tensor size and the processing resources needed. Faster processing times and a more efficient system generally follow from this, which is absolutely vital for real-time robotic applications. [12] as shown in figures 1.6 1.7.

This research intends to create a model using conditional inputs that generates accurate and efficient object detection results, especially in challenging conditions where traditional methods fail.

2

State of the Art

2.1 Overview

In this section, we will discuss about the current state-of-the-art on traditional object detection approaches, use of different encoding techniques such as visual, linguistic and contextual information which enhances accuracy and robustness of object detection, fusion methods used to combine the encoded data with the image inputs and the current use of conditional object detection and its performance when compared to the traditional methods.

2.2 Traditional Object Detection

Over the years, many new techniques have been created to make object recognition methods more accurate, faster, and more reliable. The following works give an in-depth look at important improvements in object recognition methods.

2.2.1 Literature Work

Girshick et al.'s work on R-CNN (Region-based Convolutional Neural Networks) [17] is one of the most important pieces of work in this field. This paper opened the way for region-based object recognition methods by suggesting a selective search strategy for making object proposals and then using CNN for feature extraction. This was further improved by Fast R-CNN adding to the R-CNN system by combining the feature extraction and classification steps into a single network. This made working times faster [18]. However, both R-CNN and Fast R-CNN suffered from limitations related to the reliance on external region proposal methods, which slowed down the overall detection pipeline [19].

Ren et al. did another study and came up with the Faster R-CNN model [20]. This model combined a Region Proposal Network (RPN) with the object recognition network, which made the process much faster and more accurate. Using an RPN cut down on the amount of computing power needed to make area suggestions, which meant the model could be used in real-time. Despite these improvements, Faster R-CNN still faced challenges in detecting small objects and dealing with overlapping objects due to the fixed-size anchor boxes [21].

The paper YOLO (You Only Look Once) by Redmon et al. [22] presented a new way to do object recognition by writing it as a single regression problem. This made it possible to find objects in real

time. The fact that YOLO could predict multiple bounding boxes and class percentages straight from full pictures in a single evaluation step was a big step forward in object recognition, especially when it came to speed. However, YOLO models, particularly in their earlier versions, struggled with accurately detecting small objects and exhibited poor localization accuracy, especially when objects were close together [7].

Liu et al.'s SSD (Single Shot MultiBox Detector) [8] was even better than YOLO because it used a single deep neural network to guess both the types of objects and where the bounding boxes would be at different sizes. This method made it easier for SSD to deal with things of different sizes and gave it a better mix between speed and accuracy. Nevertheless, SSD still faced limitations in detecting small objects and showed a drop in accuracy when applied to high-resolution images [23].

In recent years, transformer-based models have gained attention in object detection. Carion et al. proposed the DETR (DEtection TRansformers) framework [24], which utilized a transformer architecture to model relationships between objects directly, removing the need for many hand-designed components traditionally used in object detection pipelines. This method performed very effectively. However, DETR suffered from slow convergence during training and required a large amount of data to achieve competitive performance [25].

Object detection was cast as a keypoint detection problem in the CenterNet method [14] by Zhou et al. Finding objects was easier because CenterNet could guess where the center points would be and how big the bounding boxes would be. This meant that anchor boxes weren't needed for object detection. However, CenterNet's performance degraded when objects were densely packed, as it could confuse the center points of different objects, leading to inaccurate detections [26].

2.3 Encoding Techniques

Text encoding and label encoding are essential techniques used in machine learning and natural language processing (NLP). They turn textual information into number or category data that machine learning models can understand. Over the years, many ideas have been put forward to make these encodings faster, more accurate, and better able to understand context.

2.3.1 Literature Work

In a comprehensive research, Boykis [27] examined the concept of embeddings, highlighting its crucial role in machine learning and natural language processing. Boykis provides a comprehensive explanation of how embeddings transform categorical data into compact vectors, resulting in improved computational efficiency and model performance. The paper demonstrates the significance of vector models in several domains, ranging from language modeling to recommendation systems. It provides a robust theoretical and practical foundation for comprehending and effectively using embeddings.

One of the earliest and most straightforward approaches to text encoding is the Bag of Words (BoW) model, which was widely used for text classification tasks. BoW represents text as a collection of words, ignoring grammar and word order, and simply counts the occurrences of each word in the text [28]. Although effective in some cases, BoW suffers from the curse of dimensionality and loses contextual information because it treats all words independently.

To address the limitations of BoW, the TF-IDF (Term Frequency-Inverse Document Frequency) encoding method was introduced, which takes into account the importance of words within a corpus by assigning weights based on their frequency in individual documents versus the entire corpus [29]. While TF-IDF improves upon BoW by highlighting more informative words, it still lacks the ability to capture semantic meaning and word order.

Word2Vec, introduced by Mikolov et al., represented a significant breakthrough in text encoding [30]. Word2Vec uses neural networks to learn word embeddings, mapping words into continuous vector spaces where semantically similar words are close to each other. This method improved the ability of models to understand relationships between words, facilitating more advanced NLP tasks such as sentiment analysis and machine translation. However, Word2Vec is limited by its reliance on local context windows and its inability to handle polysemy effectively [31].

GloVe (Global Vectors for Word Representation), proposed by Pennington et al., further advanced word embeddings by combining global matrix factorization with local context window methods [32]. GloVe captures both the statistical information of words across the entire corpus and their local co-occurrence statistics, providing more accurate word representations. However, like Word2Vec, GloVe is static and does not account for word meanings that change based on context [33].

Label encoding, another method, converts categories into integers. Although label encoding is memory efficient, it introduces an artificial ordinal relationship between categories that may not exist, potentially leading to incorrect model assumptions [34].

To address these issues, embeddings for categorical data, similar to word embeddings, have been proposed, where categories are mapped into dense vector spaces. Guo and Berkhahn demonstrated that categorical embeddings significantly improve model performance for structured data by capturing hidden relationships between categories [35]. Additionally, while One-Hot Encoding and Label Encoding are effective for categorical data, they can lead to dimensionality and ordinal relationship issues that negatively impact model performance [36].

In a study by Dietterich et al. proposed a novel approach for Error-Correcting Output Codes (ECOC) [37] as an alternative to traditional encoding methods for multi-class classification problems. In this approach, each class is represented by a unique binary string (codeword), allowing for the correction of errors during classification. This work on ECOC demonstrated its potential in improving the robustness and accuracy of classifiers, particularly in noisy environments in another study conducted by Escalera et al. [38].

2.4 Fusion approaches

Fusion methods for combining encoded data and images in object detection have gained significant attention in recent years, as they offer the potential to leverage both visual and non-visual information to improve detection performance.

2.4.1 Literature Work

One of the early works in multimodal fusion was presented by Simonyan and Zisserman, who introduced the Two-Stream Convolutional Networks [39]. This approach combined spatial and temporal information from video data, using separate networks for each modality, and then fused their outputs. While primarily focused on action recognition, this work laid the groundwork for fusing different types of encoded data with images in object detection tasks.

The study conducted by George Barnum et al. (2020) investigates the advantages of early fusion in multimodal representation learning [40]. The authors propose a convolutional LSTM (C-LSTM) architecture that integrates audio and visual inputs at initial processing stages. Their findings demonstrate that early fusion enhances performance and robustness against noise, suggesting that immediate integration of modalities can significantly improve outcomes in tasks such as audio-visual speech recognition and emotion recognition.

Xu et al. proposed a novel approach by introducing a fusion network that combined RGB images and depth maps for object detection in indoor environments [41]. The network used separate convolutional neural networks (CNNs) to extract features from RGB and depth data, followed by a fusion module that merged the features before passing them to a detection head. This method improved detection accuracy in challenging indoor scenarios where depth information was crucial.

Chen et al. extended this concept by proposing the use of LiDAR data in addition to RGB images for object detection in autonomous driving applications [42]. The proposed architecture, called MV3D, used a multi-view approach where 3D point clouds from LiDAR were projected into bird's-eye view and front view maps, which were then fused with image data. This fusion of LiDAR and image data significantly improved object detection accuracy in complex driving environments.

Another significant contribution came from Ku et al., who introduced the F-PointNet model, which fused LiDAR and image data by first detecting objects in the image plane and then refining the detections using 3D point cloud data [43]. This two-stage approach allowed for more accurate localization of objects, particularly in cluttered scenes. However, the reliance on a two-stage process introduced additional computational complexity.

Zhang et al. proposed a dynamic fusion method for object detection, where the fusion process was guided by the context of the scene [44]. Their model, called Dynamic Fusion Network (DFN), used attention mechanisms to dynamically weight the contributions of different modalities based on the content of the input data. This approach improved detection accuracy in scenarios with varying lighting conditions and occlusions, but it required careful tuning of the attention mechanisms to avoid performance degradation.

A more recent development was introduced by Zhou et al., who proposed a multimodal fusion technique for 3D object detection using both images and point clouds [45]. Their model, PointFusion, utilized a two-stage approach where image features were used to guide the processing of point clouds, resulting in more accurate and robust object detection. However, the reliance on high-quality point cloud data limited the applicability of this method in environments with sparse or noisy data.

2.5 Conditional Object Detection

Conditional object detection has become a significant area of research in computer vision, focusing on enhancing object detection models by conditioning their outputs on additional inputs such as context, queries, or prior knowledge. The recent advancements in this domain have leveraged various techniques, including knowledge distillation, diffusion models, transformers, and one-shot learning, to improve detection accuracy and efficiency.

2.5.1 Literature Work

Kang et al. [46] introduced the concept of Instance-Conditional Knowledge Distillation for object detection, where the detection process is conditioned on specific instances to distill knowledge from a teacher model to a student model more effectively. This approach helps in transferring knowledge more accurately, especially in complex scenarios with multiple overlapping objects. Similarly, Chen et al. [47] proposed CamoDiffusion, a method that utilizes conditional diffusion models to detect camouflaged objects by conditioning the detection on specific visual features. This approach significantly improves the detection of objects that are difficult to distinguish from their backgrounds.

Meng et al. [48] developed the Conditional DETR model, which leverages transformers for fast training convergence by conditioning object queries on contextual information. This method enhances the efficiency of the DETR (Detection Transformer) model, enabling faster and more accurate object detection in cluttered environments. Yamada [49] explored the use of linguistic targets for object detection, where the detection process is conditioned on specific linguistic descriptions, allowing for precise detection of specified objects. This approach, however, requires the model to effectively interpret and incorporate linguistic context, which can be challenging.

In another approach, Zang et al. [50] proposed Open-Vocabulary DETR with Conditional Matching, which extends the DETR model to handle open-vocabulary tasks by conditioning on textual descriptions. This model improves detection accuracy for objects outside the training vocabulary, making it suitable for real-world applications where new object categories frequently appear. Jiang et al. [51] introduced CLIP-Count, a zero-shot object counting method that conditions on text inputs, demonstrating the potential of combining text and visual data for object detection and counting tasks.

Fu et al. [6] presented OSCD (One-Shot Conditional Detection), a framework designed to perform object detection in a one-shot setting, where the model is conditioned on a single example of the object to be detected. This approach is particularly useful in scenarios where only a few examples of the target object are available, enabling the model to generalize from limited data.

2.6 Conclusion

The analysis of state-of-the-art techniques in object recognition methods shows how far both old and new methods have come. Object recognition has been built on traditional methods like R-CNN and its variations, which have become more accurate and faster over time. But these ways still have some problems, especially when there are a lot of small objects or scenes with a lot of them.

New developments, mostly in encoding methods and multimodal fusion, look like they could help solve some of these problems. Word embeddings like Word2Vec and GloVe have made it easier to figure out what objects are used for, and fusion methods that mix data from different sources have made it easier to find things in places that are hard to understand.

The goal of conditional object detection, a new and interesting field, is to make detection better by focusing on specific objects that are important. Instance-Conditional Knowledge Distillation and Conditional DETR are two ways that use extra information, like questions and context, to make the process go faster and be more accurate. There's a chance that these methods can work around the issues with conventional ones, especially when real-time applications need to find specific items.

Overall, the improvements talked about in this chapter show that object detection technologies are always getting better. Conditional object detection is one way that these technologies can get even better at finding things in environments that are complicated and change quickly.

3

Methodology

3.1 Setup

The methodology setup for this R&D project involves the systematic collection and organization of the dataset for RoboCup@Work objects. The RoboCup@Work is a challenging international competition that aims to foster innovation in industrial robotics. For this project, the methodology was carefully designed to ensure accurate and comprehensive data collection and processing, specifically targeting the 18 objects used in the competition.

3.1.1 Object Classes and Names

The dataset comprises 18 distinct objects, each representing a different class. These objects are crucial components in the RoboCup@Work competition, and their accurate identification and classification are essential for developing reliable robotic systems. The object classes and their respective names are as follows:

1. AllenKey
2. Axis2
3. Bearing2
4. Drill
5. F20_20_B
6. F20_20_G
7. Housing
8. M20
9. M20_100
10. M30

11. Motor2
12. S40_40_B
13. S40_40_G
14. Screwdriver
15. Spacer
16. Wrench
17. Container_blue
18. Container_red

These objects are illustrated in the figures 3.1 3.2 3.3 3.4 to provide a visual reference for each class.

3.1.2 Data Collection Procedure

The data collection procedure involved the following steps:

Use of RealSense Camera:

Intel RealSense D435 camera [52] was used to capture high-quality images for the objects dataset. These cameras provided the necessary resolution and accuracy to obtain detailed visual and spatial information from multiple angles.

Labelling the Dataset:

Each image captured by the RealSense camera was carefully labeled with the corresponding object class. For annotating the dataset X-Anylabeling software [53] was used to draw bounding boxes around each object and assign the correct label. The user interface of this software is shown in figure 3.5

Labeled Dataset Images:

The final dataset consisted of images with annotated labels, indicating the exact position and class of each object within the image. These annotated labels are stored in a separate file with ".json" extension. A small snapshot of this file is shown in the figure 3.6. These labeled images are crucial for training machine learning models to recognize and classify RoboCup@Work objects accurately. The collected dataset is also a part of b-it-bots@Work [54] and is available for public use [55].

Data Preprocessing Tools:

Image processing libraries such as OpenCV and PIL were used to pre-process the images. This included resizing, normalization, and augmentation to ensure consistency and enhance the dataset's quality.

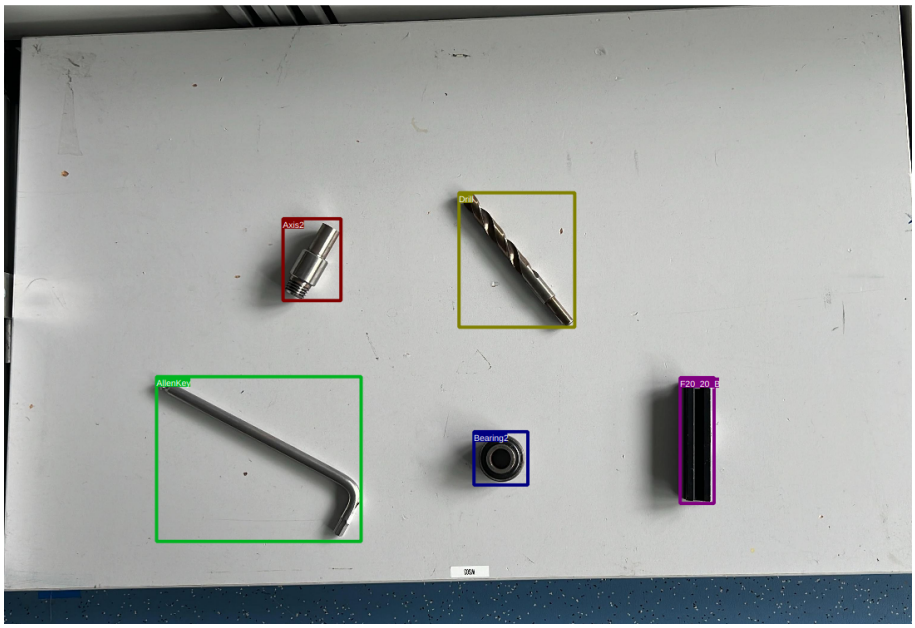

Figure 3.1: From top left: Axis2, Drill
From bottom left: AllenKey, Bearing2, F20_20_B

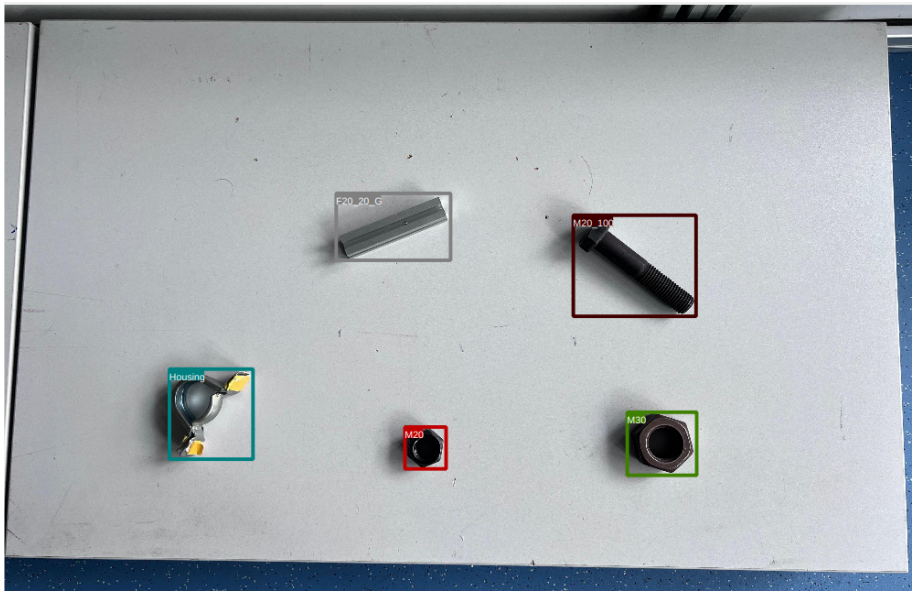

Figure 3.2: From top left: F20_20_G, M20_100
From bottom left: Housing, M20, M30

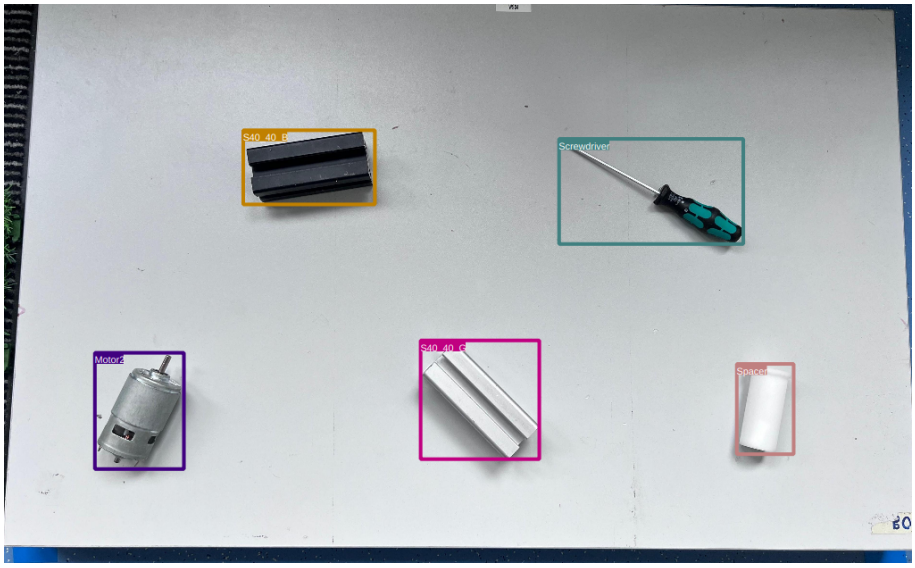

Figure 3.3: From top left: S40_40_B, Screwdriver
From bottom left: Motor2, S40_40_G, Spacer

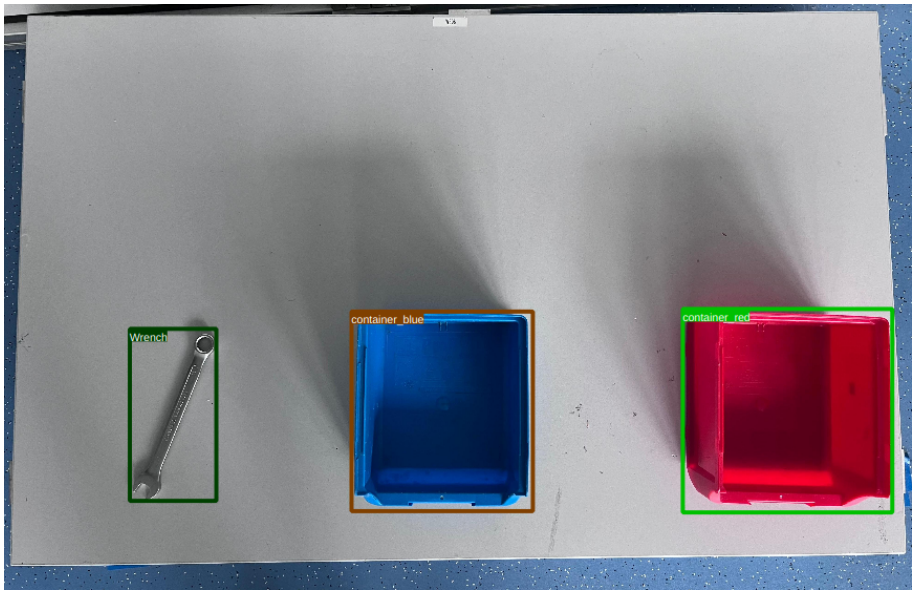

Figure 3.4: From left: Wrench, Container_Blue, Container_Red

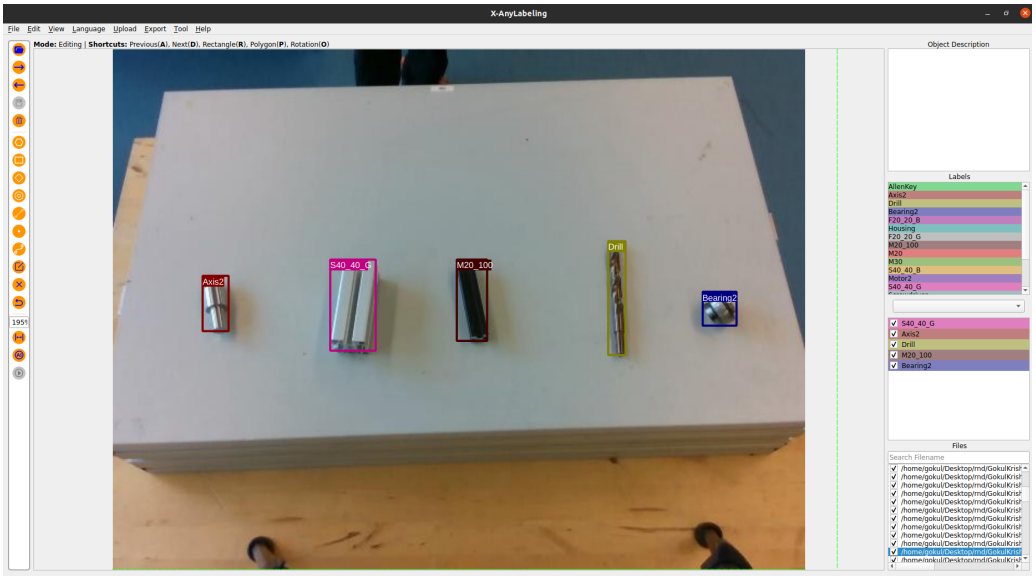

Figure 3.5: User interface of X-Anylabeling

3.1.3 Challenges and Solutions

Several challenges were encountered during the data collection process, including:

Variability in Object Appearance:

Objects with similar appearances posed a challenge in accurate classification. To address this, additional image angles and enhanced annotation techniques were employed.

Lighting and Shadows:

Variations in lighting conditions and shadows affected image quality. Controlled lighting environments were established to mitigate these issues.

Annotation Consistency:

Ensuring consistent annotations across the dataset was critical. Regular reviews and updates to the annotation guidelines helped maintain consistency.

3.2 Experimental Design

In this study, three different architectures namely ResNet50, ResNet18 and SSD are used. As a baseline the dataset is trained on all the three approaches for classification and object detection of the RoboCup@Work objects.

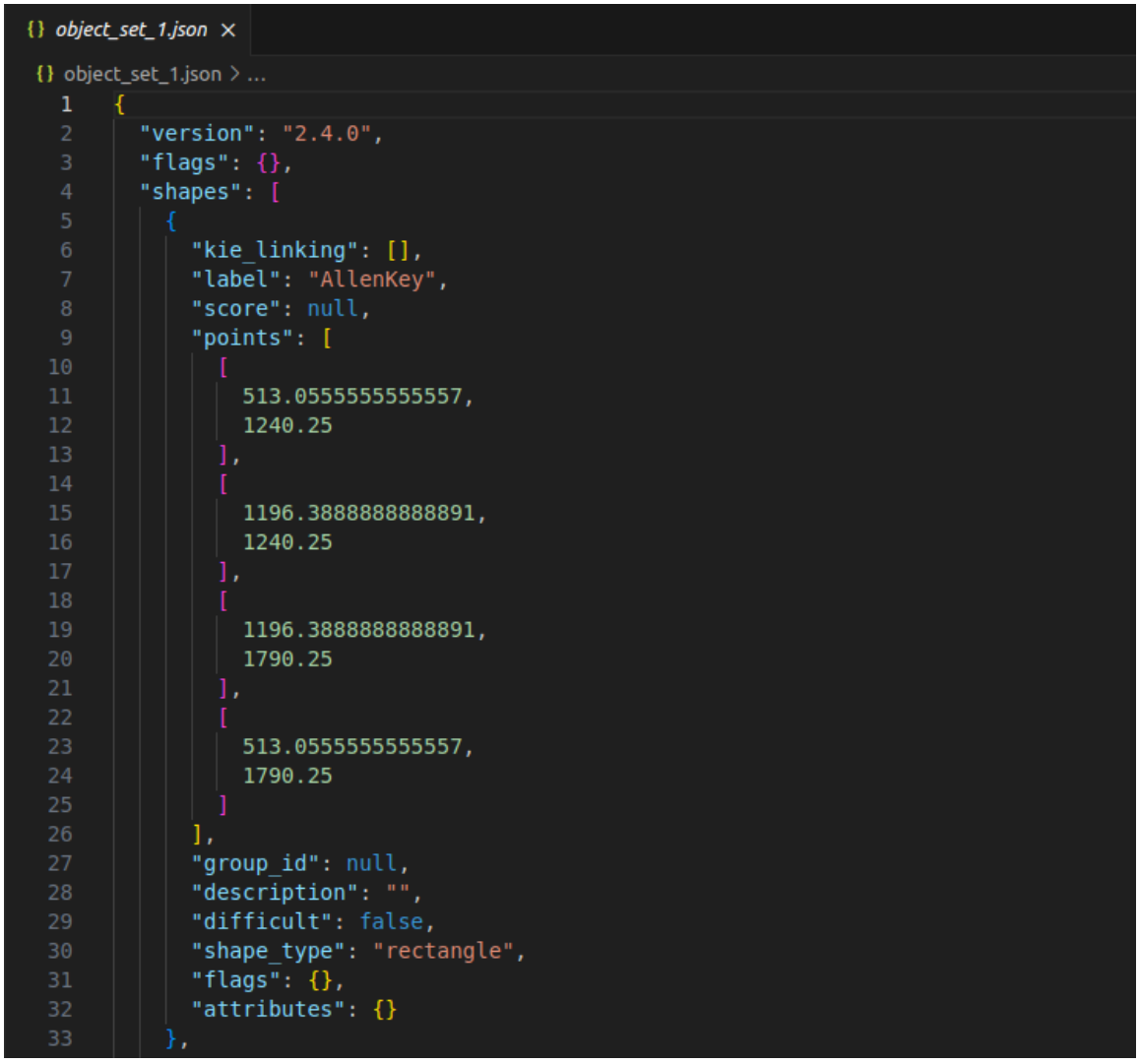

Figure 3.6: Snapshot of labeled data of an image in .json format

3.2.1 Data Preparation

Pre-processing of Image Data

- **Resizing:**
  - Images are resized to a consistent dimension (e.g., 224x224 for ResNet and SSD) to ensure uniform input size for the networks. Additionally, instead of using all the four points of the bounding box, a keypoint (x, y) is derived from these points which is the center of the rectangular box.
- **Normalization:**
  - Image pixel values are normalized to have a mean of zero and a standard deviation of one. This step is crucial for stabilizing and accelerating the training process.
- **Augmentation:**
  - Data augmentation techniques such as random cropping, horizontal flipping, rotation, scaling, and color jittering are applied to increase the diversity of the training dataset. This helps in making the model more robust to variations in the input data.

Encoding Techniques

- **One-Hot Encoding:**
  - Each categorical label is converted into a binary vector where only the index corresponding to the label is set to 1, and all other indices are set to 0.
  - This encoding scheme is commonly used in classification tasks to convert class labels into a format suitable for neural network processing.
  - The representation of the RoboCup@Work dataset objects using one-hot encoding and the plot image for the same are shown in figures3.7 3.8 respectively.
- **Error-Correcting Encoding:**
  - Categorical labels are encoded using error-correcting codes. This involves creating binary vectors with added redundancy to detect and correct errors that may occur during data transmission or processing.
  - The representation of the RoboCup@Work dataset objects using error-correcting encoding and the plot image for the same are shown in figures3.9 3.10 respectively.

```
One-Hot Encodings:
Class AllenKey       : [1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
Class Axis2          : [0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
Class Bearing2       : [0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
Class Drill          : [0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
Class F20_20_B       : [0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
Class F20_20_G       : [0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
Class Housing        : [0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
Class M20            : [0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
Class M20_100        : [0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0.]
Class M30            : [0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0.]
Class Motor2         : [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0.]
Class S40_40_B       : [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0.]
Class S40_40_G       : [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0.]
Class Screwdriver    : [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0.]
Class Spacer         : [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0.]
Class Wrench         : [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0.]
Class container_blue : [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1.]
Class container_red  : [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1.]
```

Figure 3.7: One-hot encoding codes for all object classes

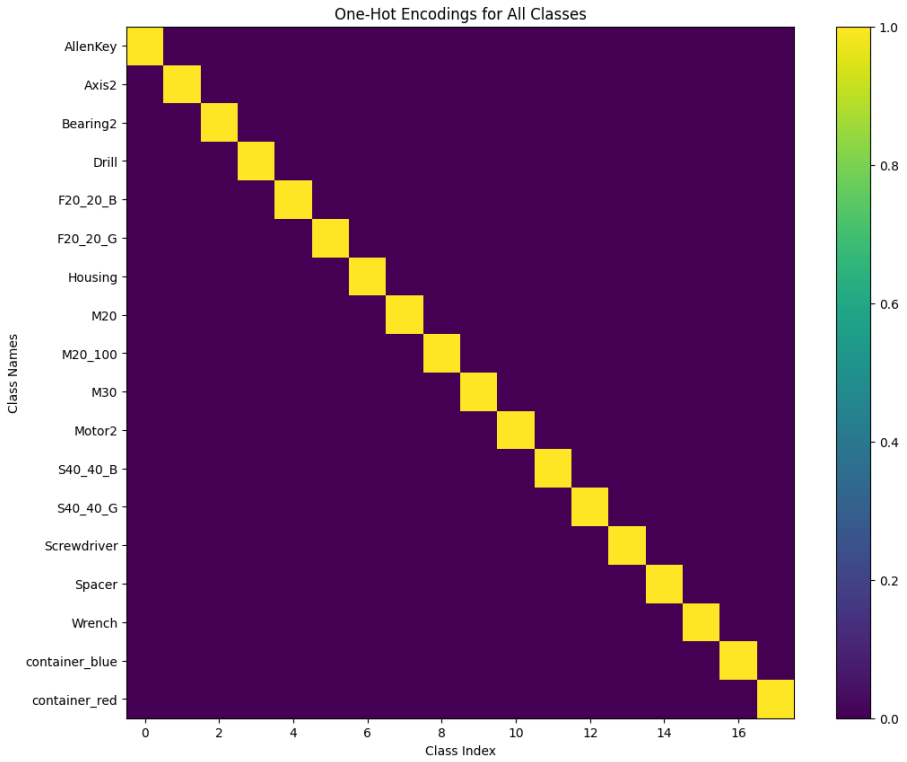

Figure 3.8: Plotting one-hot encoding codes for all object classes

```
Scipy Hadamard Matrix Shape:  (16, 16)
Error Correcting Output Codes:
Class AllenKey        : [ 1.  1.  1. -1. -1. -1.  1. -1. -1.  1.  1.  1. -1.  1. -1. -1.]
Class Axis2           : [-1. -1.  1. -1.  1. -1. -1.  1. -1.  1.  1.  1. -1. -1.  1.  1.]
Class Bearing2        : [ 1.  1.  1. -1. -1.  1. -1.  1. -1. -1. -1.  1.  1. -1.  1. -1.]
Class Drill           : [-1.  1.  1. -1.  1. -1. -1. -1.  1. -1.  1. -1.  1.  1.  1. -1.]
Class F20_20_B        : [ 1. -1.  1.  1.  1.  1.  1. -1. -1. -1.  1. -1. -1. -1.  1. -1.]
Class F20_20_G        : [1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
Class Housing         : [ 1. -1.  1. -1. -1. -1.  1.  1.  1. -1.  1. -1.  1. -1. -1.  1.]
Class M20             : [-1.  1.  1.  1. -1.  1. -1.  1. -1. -1.  1. -1. -1.  1. -1.  1.]
Class M20_100         : [-1. -1.  1. -1.  1.  1.  1. -1. -1. -1. -1.  1.  1.  1. -1.  1.]
Class M30             : [-1. -1.  1.  1. -1.  1. -1. -1.  1.  1.  1.  1.  1. -1. -1. -1.]
Class Motor2          : [-1.  1.  1.  1. -1. -1.  1. -1. -1.  1. -1. -1.  1. -1.  1.  1.]
Class S40_40_B        : [ 1.  1.  1.  1.  1. -1. -1. -1.  1. -1. -1.  1. -1. -1. -1.  1.]
Class S40_40_G        : [ 1. -1.  1.  1.  1. -1. -1.  1. -1.  1. -1. -1.  1.  1. -1. -1.]
Class Screwdriver     : [-1. -1.  1.  1. -1. -1.  1.  1.  1. -1. -1.  1. -1.  1.  1. -1.]
Class Spacer          : [-1.  1.  1. -1.  1.  1.  1.  1.  1.  1. -1. -1. -1. -1. -1. -1.]
Class Wrench          : [ 1. -1.  1. -1. -1.  1. -1. -1.  1.  1. -1. -1. -1.  1.  1.  1.]
Class container_blue  : [-1. -1. -1. -1.  1. -1.  1. -1. -1.  1. -1.  1.  1. -1.  1. -1.]
Class container_red   : [-1.  1.  1. -1. -1. -1.  1.  1. -1.  1.  1.  1. -1. -1. -1. -1.]
```

Figure 3.9: Error-correcting encoding codes for all object classes

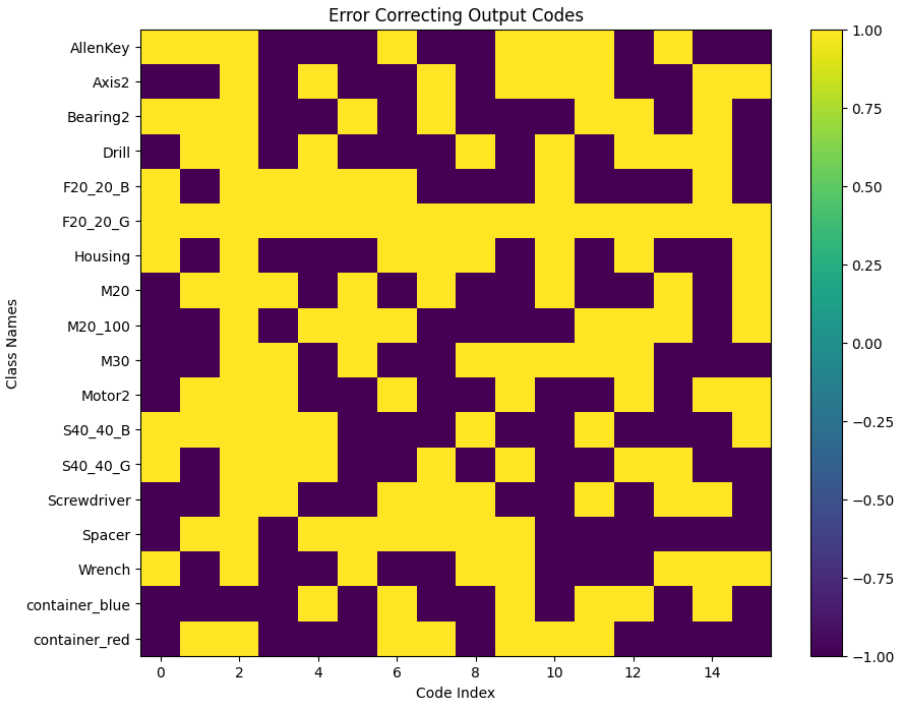

Figure 3.10: Plotting error-correcting encoding codes for all object classes

3.2.2 Base Networks

ResNet50:

ResNet50 is a deep convolutional neural network consisting of 50 layers. The architecture addresses the vanishing gradient problem through the use of residual learning. The core idea behind residual learning is to add shortcut connections that bypass one or more layers, enabling the network to learn identity mappings. This significantly improves the training of very deep networks.

- **Initial Layers:** ResNet50 starts with an initial convolutional layer followed by batch normalization and ReLU activation. This is followed by a max-pooling layer that reduces the spatial dimensions.
- **Residual Blocks:** The network is composed of several residual blocks, each containing three convolutional layers. These layers are followed by batch normalization and ReLU activation. The input to each residual block is added to the output of the block via the shortcut connection, facilitating easier gradient flow during backpropagation.
- **Final Layers:** After passing through multiple residual blocks, the network has an average pooling layer that reduces the feature map to a fixed size, followed by fully connected layers that output the final class predictions.

The depth of ResNet50 allows it to capture high-level features, making it suitable for complex image recognition tasks.

ResNet18:

ResNet18, a smaller variant of the ResNet architecture, consists of 18 layers. It maintains the same principles of residual learning but with fewer layers, making it more computationally efficient. This architecture is beneficial when computational resources are limited or when faster training times are required.

- **Initial Layers:** Similar to ResNet50, ResNet18 starts with a convolutional layer, batch normalization, and ReLU activation, followed by max-pooling.
- **Residual Blocks:** The network includes several residual blocks, each containing two convolutional layers. The shortcut connections are applied in the same manner as in ResNet50.
- **Final Layers:** After processing through the residual blocks, the network ends with an average pooling layer and fully connected layers for class prediction.

Despite its reduced depth, ResNet18 still benefits from the advantages of residual learning, making it effective for various image classification tasks.

SSD (Single Shot MultiBox Detector):

The Single Shot MultiBox Detector (SSD) [8] is an efficient and powerful object detection model that predicts object classes and locations in a single forward pass. SSD achieves high detection accuracy through the use of multiple convolutional feature layers and default bounding boxes of different aspect ratios.

- **Base Network:** The SSD architecture typically employs a pre-trained network such as VGG16 or ResNet as the base for feature extraction. This base network is followed by additional convolutional layers.
- **Convolutional Feature Layers:** SSD adds several convolutional layers on top of the base network, each producing a set of feature maps at different scales. These layers allow the model to detect objects of various sizes.
- **Default Boxes and Predictions:** For each feature map cell, SSD defines a set of default boxes with different aspect ratios and scales. The model predicts both the class scores and the offsets for these default boxes. During inference, non-maximum suppression is used to filter out redundant detections and refine the final predictions.

SSD's architecture, with its efficient single-shot approach, is well-suited for real-time object detection tasks.

3.2.3 Image and Encoded Label Fusion Techniques

Before the data is passed on to the model network for training, both the image and the encoded label are fused into a single feature data. The fusion methods used in this project are discussed below in detail.

Concatenation:

Concatenation is a straightforward fusion method where data arrays are joined along a specified dimension. When applied to image features and encoded labels, this method stacks the data arrays, creating a larger combined array.

- **Combining Data Arrays:** The image features and encoded labels are concatenated along a chosen axis, forming a single array that contains both sets of information.
- **Dimensionality:** The resulting array's dimensionality is the sum of the dimensions of the individual arrays along the concatenation axis.

Concatenation is effective for combining heterogeneous data sources, ensuring that both image features and encoded labels are available for subsequent processing.

Addition:

Addition is a fusion method where corresponding elements of the data arrays are summed. This method is simple and can be applied when the data arrays have the same dimensions.

- **Element-wise Summation:** The image features and encoded labels are added element-wise, resulting in a single array that combines information from both sources.
- **Dimensionality:** The dimensionality of the resulting array is the same as the original data arrays.

Addition is useful for integrating complementary data, enhancing the combined representation's informative value.

Multiplication:

Multiplication involves element-wise multiplication of data arrays. This method emphasizes interactions between different data sources by amplifying features present in both arrays.

- **Element-wise Multiplication:** The image features and encoded labels are multiplied element-wise, producing a single array that captures the interactions between the data sources.
- **Dimensionality:** he dimensionality of the resulting array matches that of the original data arrays.

Multiplication can highlight important correlations between image features and encoded labels, providing a rich combined representation.

3.2.4 Altering Network Architecture for Conditional Object Detection

In order to effectively utilize both image features and encoded labels for conditional object detection, it is crucial to adapt the network architecture to handle and process this combined data. This involves designing specific layers and methods to fuse the data, adapt the network to the increased complexity, and ensure accurate keypoint prediction for object detection.

Input Layer:

- **Separate Input Layers:**
  - **Image Input:** The image data is typically a three-channel RGB image with dimensions $3XHXW$(three color channels, height and width).
  - **Encoded Label Input:** The encoded labels, represented as a single-channel vector with dimensions $1XHXW$, are concatenated along the channel dimension to the image data.
  - **Combined Input:** When concatenating, the combined input will have dimensions $4XHXW$ for concatenation fusion method, where the fourth channel is the encoded label and $3XHXW$ for addition and multiplication fusion methods.

- **Preprocessing:**
  - Both the image data and encoded labels need to be preprocessed. For images, this might include normalization and resizing, while encoded labels might require normalization or scaling to ensure they are in a compatible range.

Fusion Layer:

In this layer the actual combination of image features and encoded label is performed. The chosen fusion method (concatenation, addition, or multiplication) will affect how these inputs are fused.

- **Concatenation:**
  - The image and encoded label data are concatenated along the channel dimension resulting in the dimensions $4XHXW$.
- **Addition:**
  - Both inputs must have the same dimensions.
  - The image data and encoded labels are added element-wise resulting in the dimensions $3XHXW$.
- **Multiplication:**
  - Both inputs must have the same dimensions.
  - The image data and encoded labels are multiplied element-wise resulting in the dimensions $3XHXW$.

Intermediate Layers:

The intermediate layers must be adapted to handle the increased size and complexity of the combined data. This involves making several changes to the architecture:

- **Convolutional Layers:**
  - **Increased Filters:** Increase the number of filters in the initial convolutional layers to accommodate the additional channel(s) introduced by the fusion layer.
- **Pooling Layers:**
  - Pooling layers (e.g., max-pooling or average-pooling) reduce the spatial dimensions of the feature maps. Ensure these layers are compatible with the dimensions of the fused data.
- **Fully Connected Layers:**
  - Increase the number of units in the fully connected layers to handle the richer feature set produced by the combined data.

- **Batch Normalization and Activation Layers:**
  - Use batch normalization to stabilize training and improve convergence.
  - Apply appropriate activation functions (e.g., ReLU) to introduce non-linearity and enhance model performance.

Keypoint Prediction:

For keypoint prediction, we focus on predicting the center of the bounding box rather than the full bounding box coordinates. This involves adjusting the output layers and the loss function.

- **Output Representation:**
  - The output layer should predict the coordinates of the keypoint (center of the bounding box).
  - For each object, the network outputs a set of coordinates (x, y) representing the center point.
- **Loss Function:**
  - Use a regression loss function (e.g., mean squared error and laplace) to minimize the distance between the predicted keypoints and the true keypoints.

Output Layer:

The final output layer of the network must be designed to predict the presence and class of objects based on the combined input data. This involves several considerations:

- **Class Prediction:**
  - Use a softmax activation function to output class probabilities for each object.
  - The number of output neurons should match the number of classes in the dataset.
- **Keypoint Coordinates:**
  - Use a linear activation function for the keypoint coordinate predictions.
  - The output layer should have two neurons per predicted object (one for x and one for y coordinate).
- **Combining Predictions:**
  - The final output is a combination of class predictions and keypoint coordinates.

3.2.5 Training and Evaluation

Loss Functions:

- **Cross-Entropy Loss:** Used for classification, this loss function measures the difference between predicted class probabilities and true labels.
- **Mean Squared Error (MSE) Loss:** Used for regression tasks, it measures the average of the squares of the errors between the predicted values and the true values. MSE is particularly sensitive to outliers, which can be both an advantage and a disadvantage depending on the application.

  $\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$
- **Laplace Loss:** The Laplace loss function, also known as the L1 loss or mean absolute error, is used for regression tasks where robustness to outliers is desired. It measures the absolute differences between the predicted values and the true values.

  $\text{Laplace Loss} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$

Optimization:

- **Algorithm:** The network is trained using optimization algorithms like Adam or SGD, which adjust the weights to minimize the loss functions.
- **Learning Rate Scheduling:** Techniques like reducing the learning rate on plateau are employed to fine-tune the learning process and achieve better performance.
- **Regularization:** Dropout, weight decay, and batch normalization are used to prevent overfitting and ensure the model generalizes well to unseen data.

The parameters used for the training of the model are mentioned in the table 3.1

Parameter	Value
`epochs`	100/**250**
`learning rate`	1e-3/**1e-4**
`image size`	[**224, 224**]
`batch`	32
`stride`	**2**

Table 3.1: Training Parameters for ResNet50 and ResNet18 backbones

Evaluation:

The trained model is evaluated on a test set, using metrics like speed(ms), accuracy(Intersection over Union(IoU)) mean Average Precision (mAP) to assess performance. These metrics provide insight into the model's accuracy and effectiveness in detecting objects and the performance comparison table is shown in 3.2

Table 3.2: Performance comparison of traditional methods vs conditional approach using different detection and fusion methods

Architecture	**Detection**	**Encoding**	**Fusion**	**Speed(ms)**		**Accuracy(IoU)**		**mAP**	
				100 Epo	**250 Epo**	**100 Epo**	**250 Epo**	**100 Epo**	**250 Epo**
Base model	Traditional								
	Conditional	One-hot	Concatenation Addition Multiplication						
		Error-correcting	Concatenation Addition Multiplication						

4

Evaluation and Results

4.1 Experiment Description

The experiment description is done in very detail in the previous chapter, Methodology. This previous chapter has many important parts that are needed to understand how it works. This includes a full explanation of the steps that were taken to prepare the data, such as how the data was resized, made standard, and added to.

4.2 Experimental Setup

The previous chapter, Methodology also covers the designs that were employed, including ResNet50, ResNet18, and SSD, are also covered, along with how the models were improved and updated before to testing. It also describes how the network structure was altered to support conditional object detection and how image data and encoded labels were combined. Along with the loss functions and improvement strategies, it also describes how the testing and training processes operate. Also, the experiment in the same base network is compared with respect to different regression loss function namely MSE loss (Mean Squared Error Loss) and Laplace loss functions.

4.3 Results

The experiment is done for two variants of epoch values 100 and 250 respectively and for two regression loss functions MSE loss and Laplace loss functions for ResNet18, ResNet50 and SSD networks.

4.3.1 Training Plots for Different Encoding and Fusion Methods

ResNet18 Network Plots

Using ResNet18 network, for one-hot encoding with MSE loss, the keypoint loss over both the epochs are almost similar for all three fusion methods (concatenation, addition and multiplication) with loss decreasing steadily with very minute fluctuations. After a certain number of epoch values the loss remains almost constant with tendency of no further learning. While the classification loss with Cross-Entropy Loss decreases with heavy fluctuations while training over both the epochs.

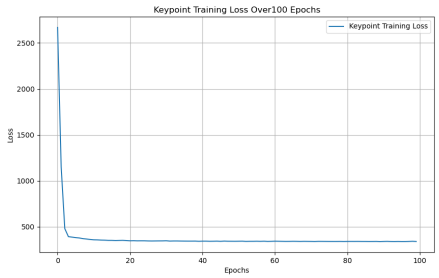

Figure 4.1: Plot for ResNet18 keypoint training loss with MSE loss over 100 epochs for one-hot encoding using concatenation

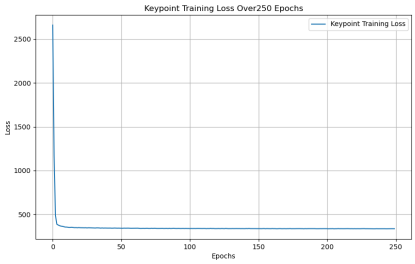

Figure 4.2: Plot for ResNet18 keypoint training loss with MSE loss over 250 epochs for one-hot encoding using concatenation

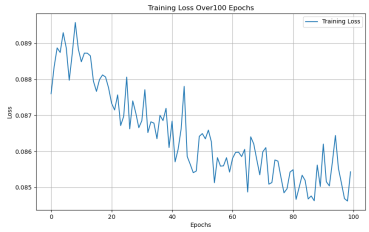

Figure 4.3: Plot for ResNet18 classification training loss with MSE loss over 100 epochs for one-hot encoding using concatenation

Figure 4.4: Plot for ResNet18 classification training loss with MSE loss over 250 epochs for one-hot encoding using concatenation

Similarly, for error-correcting encoding with MSE loss, the keypoints loss over both the epochs are similar for all three fusion methods (concatenation, addition and multiplication) with loss decreasing to a certain value and then slowing down for some iterations until a stable point i.e around 50-60 epoch values and then the loss curve remains stable with minute fluctuations. While the classification loss with Cross-Entropy Loss decreases with way less fluctuations when compared with one-hot encoding while training over both the epochs.

For one-hot encoding with Laplace loss, the keypoint loss over both the epochs are almost similar for all three fusion methods (concatenation, addition and multiplication) with loss having fluctuations all during the training and not attaining a stable curve. While the classification loss with Cross-Entropy Loss also have pass through with heavy fluctuations while training and does not attain any stable curve over both the epochs. The case is very similar for error-correcting encoding with Laplace loss for both keypoint and classification.

The plots for training of dataset for one-hot and error-encoding using both MSE loss and Laplace loss for both keypoint loss and classification loss in ResNet18 network are shown from figures 4.1 to 4.48

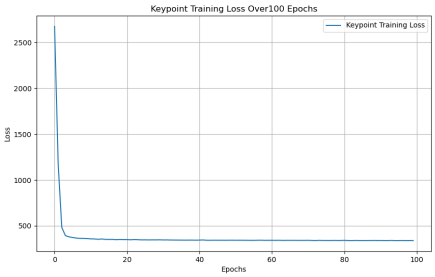

Figure 4.5: Plot for ResNet18 keypoint training loss with MSE loss over 100 epochs for one-hot encoding using addition

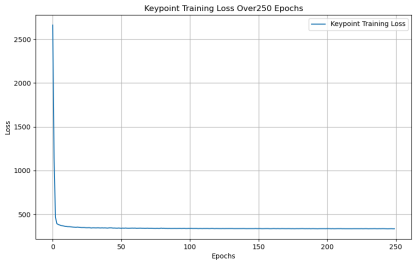

Figure 4.6: Plot for ResNet18 keypoint training loss with MSE loss over 250 epochs for one-hot encoding using addition

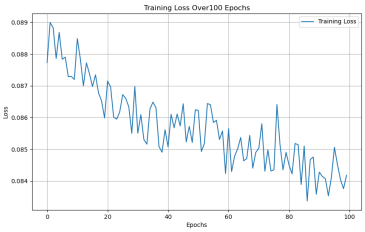

Figure 4.7: Plot for ResNet18 classification training loss with MSE loss over 100 epochs for one-hot encoding using addition

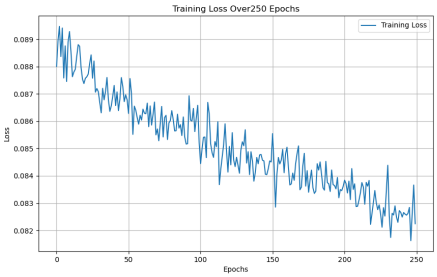

Figure 4.8: Plot for ResNet18 classification training loss with MSE loss over 250 epochs for one-hot encoding using addition

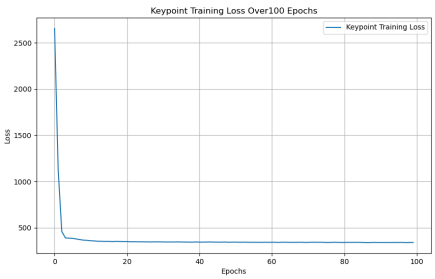

Figure 4.9: Plot for ResNet18 keypoint training loss with MSE loss over 100 epochs for one-hot encoding using multiplication

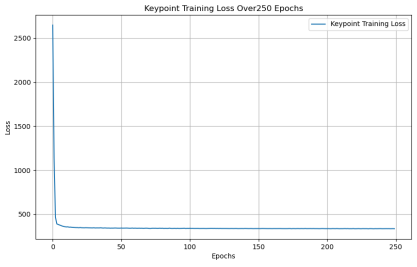

Figure 4.10: Plot for ResNet18 keypoint training loss with MSE loss over 250 epochs for one-hot encoding using multiplication

Figure 4.11: Plot for ResNet18 classification training loss with MSE loss over 100 epochs for one-hot encoding using multiplication

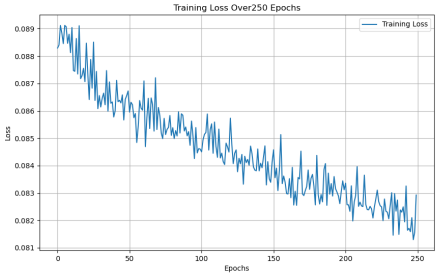

Figure 4.12: Plot for ResNet18 classification training loss with MSE loss over 250 epochs for one-hot encoding using multiplication

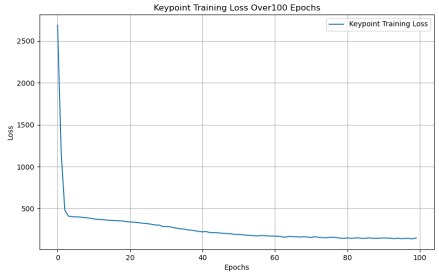

Figure 4.13: Plot for ResNet18 keypoint training loss with MSE loss over 100 epochs for error-correcting encoding using concatenation

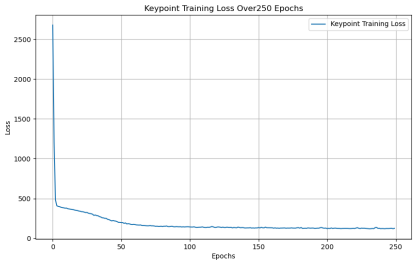

Figure 4.14: Plot for ResNet18 keypoint training loss with MSE loss over 250 epochs for error-correcting encoding using concatenation

Figure 4.15: Plot for ResNet18 classification training loss with MSE loss over 100 epochs for error-correcting encoding using concatenation

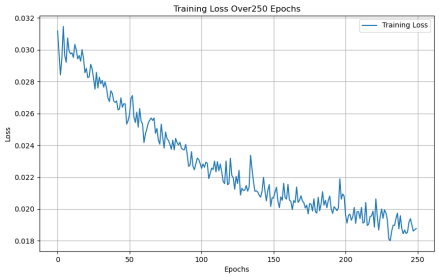

Figure 4.16: Plot for ResNet18 classification training loss with MSE loss over 250 epochs for error-correcting encoding using concatenation

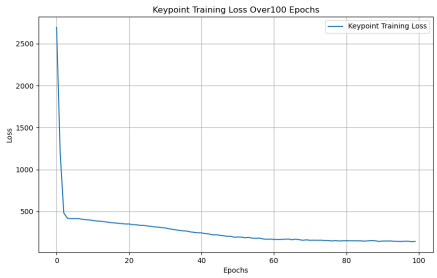

Figure 4.17: Plot for ResNet18 keypoint training loss with MSE loss over 100 epochs for error-correcting encoding using addition

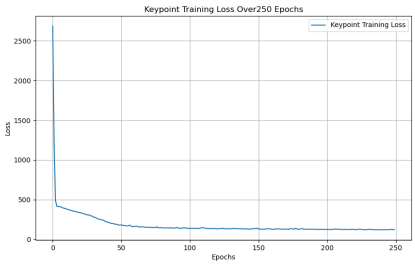

Figure 4.18: Plot for ResNet18 keypoint training loss with MSE loss over 250 epochs for error-correcting encoding using addition

Figure 4.19: Plot for ResNet18 classification training loss with MSE loss over 100 epochs for error-correcting encoding using addition

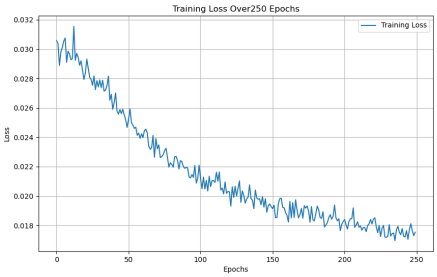

Figure 4.20: Plot for ResNet18 classification training loss with MSE loss over 250 epochs for error-correcting encoding using addition

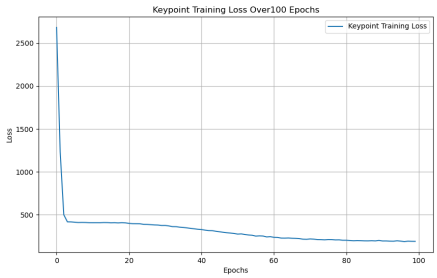

Figure 4.21: Plot for ResNet18 keypoint training loss with MSE loss over 100 epochs for error-correcting encoding using multiplication

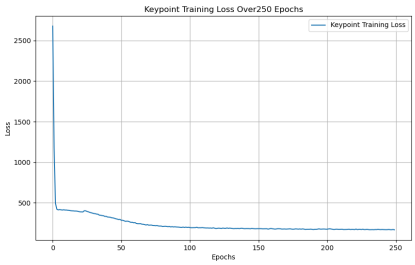

Figure 4.22: Plot for ResNet18 keypoint training loss with MSE loss over 250 epochs for error-correcting encoding using multiplication

Figure 4.23: Plot for ResNet18 classification training loss with MSE loss over 100 epochs for error-correcting encoding using multiplication

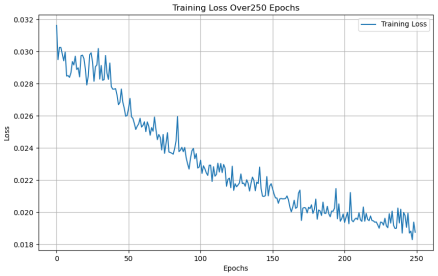

Figure 4.24: Plot for ResNet18 classification training loss with MSE loss over 250 epochs for error-correcting encoding using multiplication

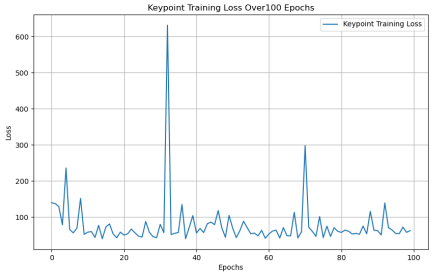

Figure 4.25: Plot for ResNet18 keypoint training loss with Laplace loss over 100 epochs for one-hot encoding using concatenation

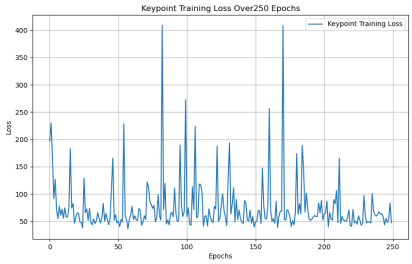

Figure 4.26: Plot for ResNet18 keypoint training loss with Laplace loss over 250 epochs for one-hot encoding using concatenation

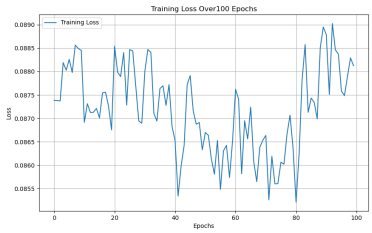

Figure 4.27: Plot for ResNet18 classification training loss with Laplace loss over 100 epochs for one-hot encoding using concatenation

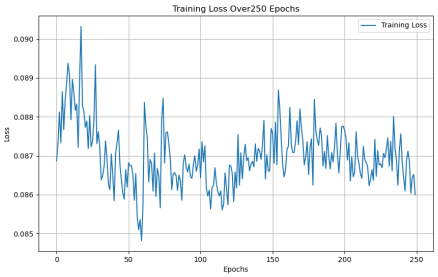

Figure 4.28: Plot for ResNet18 classification training loss with Laplace loss over 250 epochs for one-hot encoding using concatenation

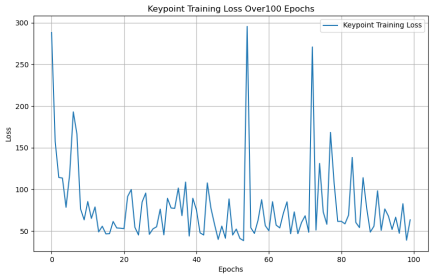

Figure 4.29: Plot for ResNet18 keypoint training loss with Laplace loss over 100 epochs for one-hot encoding using addition

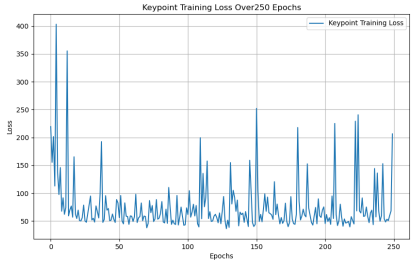

Figure 4.30: Plot for ResNet18 keypoint training loss with Laplace loss over 250 epochs for one-hot encoding using addition

Figure 4.31: Plot for ResNet18 classification training loss with Laplace loss over 100 epochs for one-hot encoding using addition

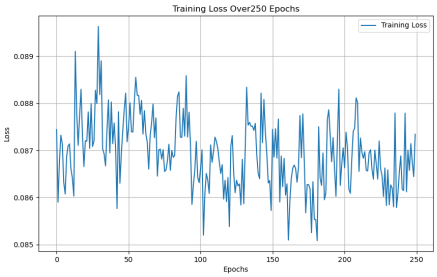

Figure 4.32: Plot for ResNet18 classification training loss with Laplace loss over 250 epochs for one-hot encoding using addition

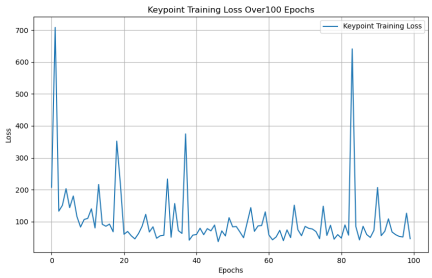

Figure 4.33: Plot for ResNet18 keypoint training loss with Laplace loss over 100 epochs for one-hot encoding using multiplication

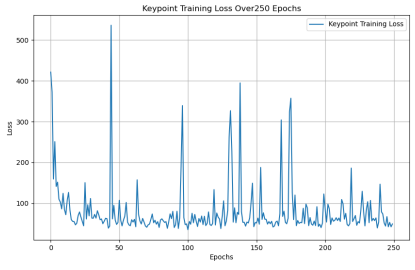

Figure 4.34: Plot for ResNet18 keypoint training loss with Laplace loss over 250 epochs for one-hot encoding using multiplication

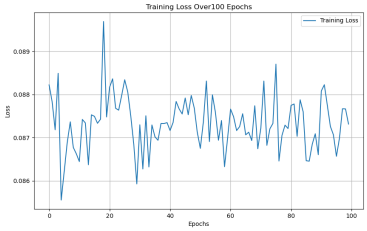

Figure 4.35: Plot for ResNet18 classification training loss with Laplace loss over 100 epochs for one-hot encoding using multiplication

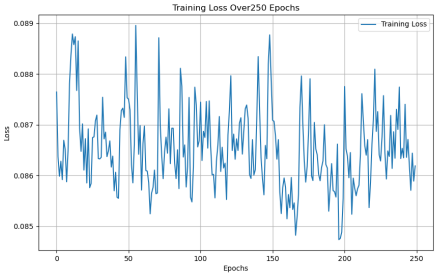

Figure 4.36: Plot for ResNet18 classification training loss with Laplace loss over 250 epochs for one-hot encoding using multiplication

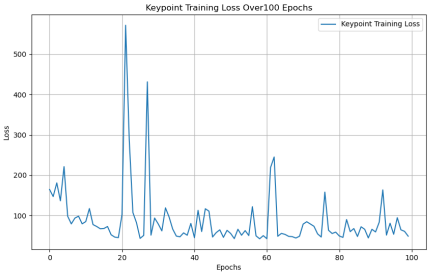

Figure 4.37: Plot for ResNet18 keypoint training loss with Laplace loss over 100 epochs for error-correcting encoding using concatenation

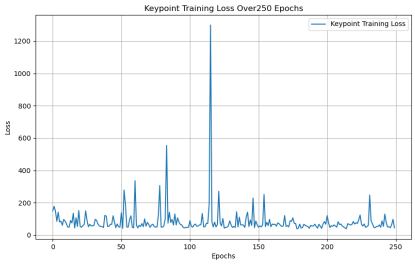

Figure 4.38: Plot for ResNet18 keypoint training loss with Laplace loss over 250 epochs for error-correcting encoding using concatenation

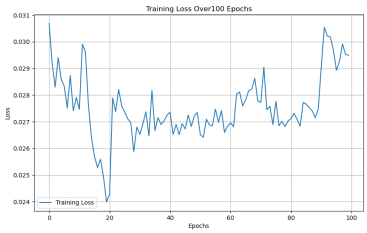

Figure 4.39: Plot for ResNet18 classification training loss with Laplace loss over 100 epochs for error-correcting encoding using concatenation

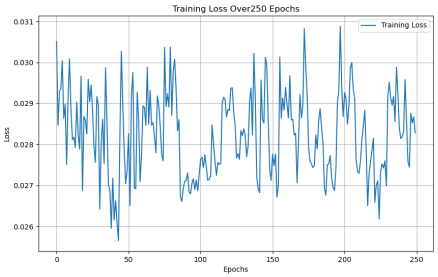

Figure 4.40: Plot for ResNet18 classification training loss with Laplace loss over 250 epochs for error-correcting encoding using concatenation

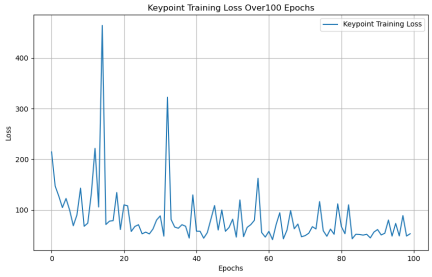

Figure 4.41: Plot for ResNet18 keypoint training loss with Laplace loss over 100 epochs for error-correcting encoding using addition

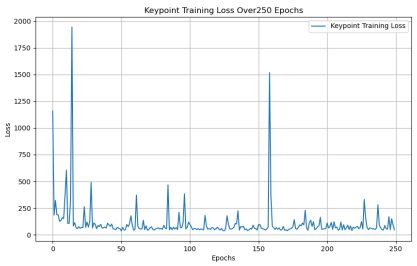

Figure 4.42: Plot for ResNet18 keypoint training loss with Laplace loss over 250 epochs for error-correcting encoding using addition

Figure 4.43: Plot for ResNet18 classification training loss with Laplace loss over 100 epochs for error-correcting encoding using addition

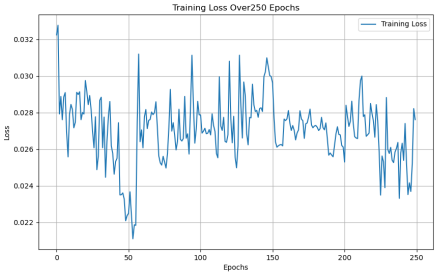

Figure 4.44: Plot for ResNet18 classification training loss with Laplace loss over 250 epochs for error-correcting encoding using addition

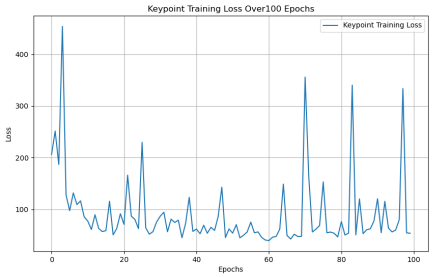

Figure 4.45: Plot for ResNet18 keypoint training loss with Laplace loss over 100 epochs for error-correcting encoding using multiplication

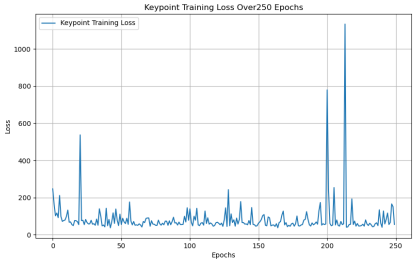

Figure 4.46: Plot for ResNet18 keypoint training loss with Laplace loss over 250 epochs for error-correcting encoding using multiplication

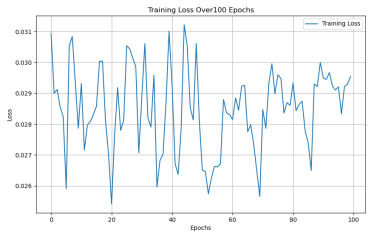

Figure 4.47: Plot for ResNet18 classification training loss with Laplace loss over 100 epochs for error-correcting encoding using multiplication

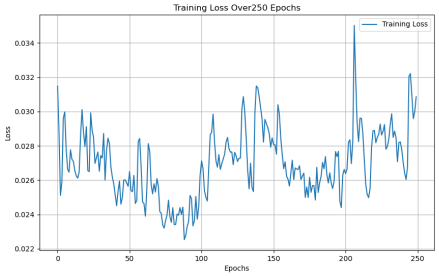

Figure 4.48: Plot for ResNet18 classification training loss with Laplace loss over 250 epochs for error-correcting encoding using multiplication

ResNet50 Network Plots

Using ResNet50 network, for one-hot encoding with MSE loss, the keypoint loss over both the epochs are almost similar for all three fusion methods (concatenation, addition and multiplication) with loss decreasing steadily with very minute fluctuations. After a certain number of epoch values the loss remains almost constant with tendency of no further learning similar to what was observed in ResNet18 network. While the classification loss with Cross-Entropy Loss decreases with fluctuations while training over both the epochs and less fluctuations when compared with ResNet18 network.

Similarly, for error-correcting encoding with MSE loss, the keypoints loss over both the epochs are similar for all three fusion methods (concatenation, addition and multiplication) with loss decreasing to a certain value and then slowing down for some iterations until a stable point i.e around 50-60 epoch values and then the loss curve remains stable with minute fluctuations similar to what was observed in ResNet18 network. While the classification loss with Cross-Entropy Loss decreases with way less fluctuations when compared with one-hot encoding while training over both the epochs. Classification loss for this network is similar to what was observed in ResNet18 network.

For one-hot encoding with Laplace loss, the keypoint loss over both the epochs are almost similar for all three fusion methods (concatenation, addition and multiplication) with loss having fluctuations all during the training and not attaining a stable curve. While the classification loss with Cross-Entropy Loss also have pass through with heavy fluctuations while training and does not attain any stable curve over both the epochs. The case is very similar for error-correcting encoding with Laplace loss for both keypoint and classification. In this case, both the keypoint loss and classification loss trend is similar to what was observed in ResNet18 network.

The plots for these training of dataset for one-hot and error-encoding using both MSE loss and Laplace loss for both keypoint loss and classification loss in ResNet50 network are shown from figures 4.49 to 4.96

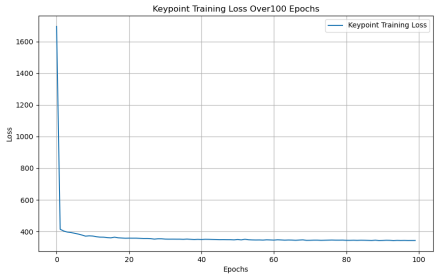

Figure 4.49: Plot for ResNet50 keypoint training loss with MSE loss over 100 epochs for one-hot encoding using concatenation

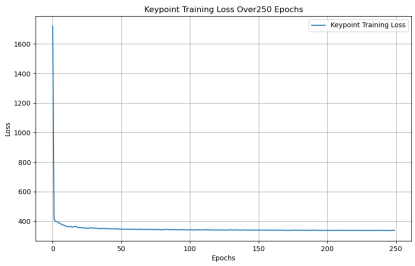

Figure 4.50: Plot for ResNet50 keypoint training loss with MSE loss over 250 epochs for one-hot encoding using concatenation

Figure 4.51: Plot for ResNet50 classification training loss with MSE loss over 100 epochs for one-hot encoding using concatenation

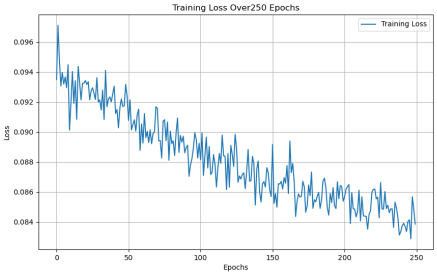

Figure 4.52: Plot for ResNet50 classification training loss with MSE loss over 250 epochs for one-hot encoding using concatenation

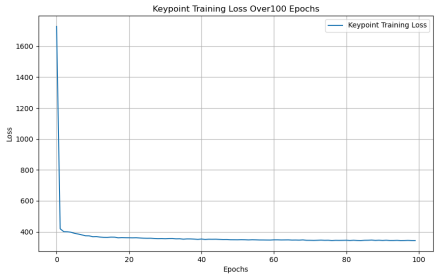

Figure 4.53: Plot for ResNet50 keypoint training loss with MSE loss over 100 epochs for one-hot encoding using addition

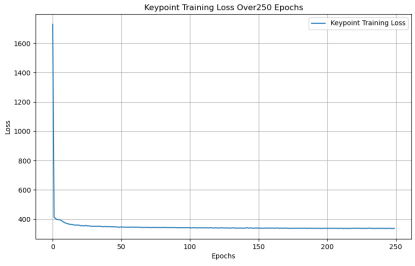

Figure 4.54: Plot for ResNet50 keypoint training loss with MSE loss over 250 epochs for one-hot encoding using addition

Figure 4.55: Plot for ResNet50 classification training loss with MSE loss over 100 epochs for one-hot encoding using addition

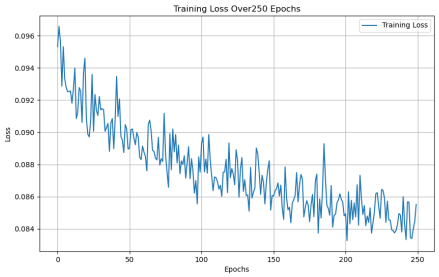

Figure 4.56: Plot for ResNet50 classification training loss with MSE loss over 250 epochs for one-hot encoding using addition

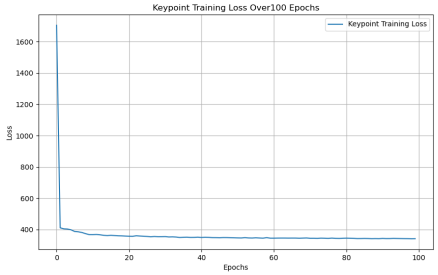

Figure 4.57: Plot for ResNet50 keypoint training loss with MSE loss over 100 epochs for one-hot encoding using multiplication

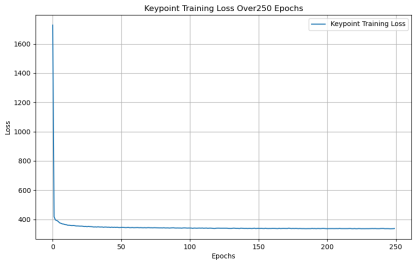

Figure 4.58: Plot for ResNet50 keypoint training loss with MSE loss over 250 epochs for one-hot encoding using multiplication

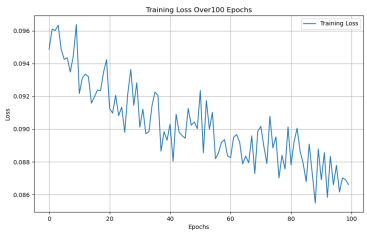

Figure 4.59: Plot for ResNet50 classification training loss with MSE loss over 100 epochs for one-hot encoding using multiplication

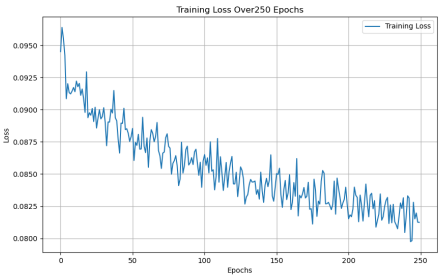

Figure 4.60: Plot for ResNet50 classification training loss with MSE loss over 250 epochs for one-hot encoding using multiplication

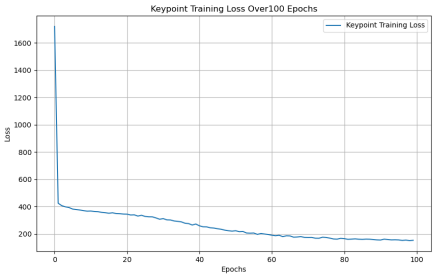

Figure 4.61: Plot for ResNet50 keypoint training loss with MSE loss over 100 epochs for error-correcting encoding using concatenation

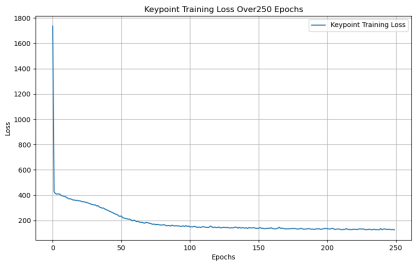

Figure 4.62: Plot for ResNet50 keypoint training loss with MSE loss over 250 epochs for error-correcting encoding using concatenation

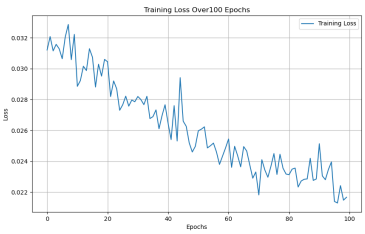

Figure 4.63: Plot for ResNet50 classification training loss with MSE loss over 100 epochs for error-correcting encoding using concatenation

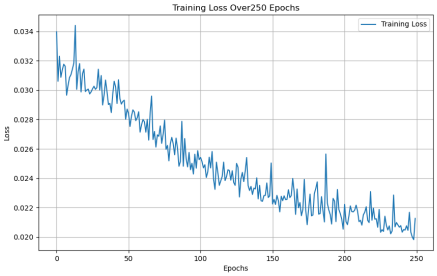

Figure 4.64: Plot for ResNet50 classification training loss with MSE loss over 250 epochs for error-correcting encoding using concatenation

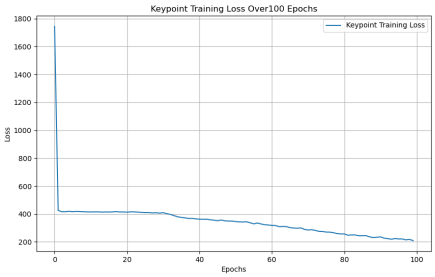

Figure 4.65: Plot for ResNet50 keypoint training loss with MSE loss over 100 epochs for error-correcting encoding using addition

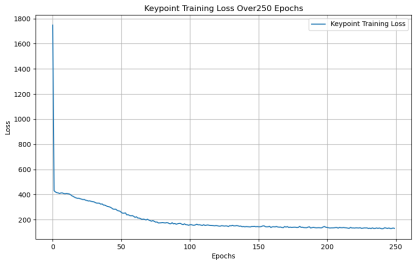

Figure 4.66: Plot for ResNet50 keypoint training loss with MSE loss over 250 epochs for error-correcting encoding using addition

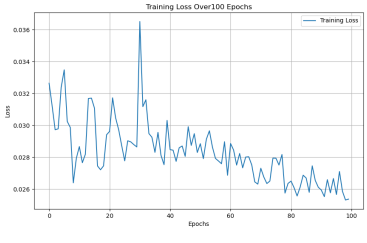

Figure 4.67: Plot for ResNet50 classification training loss with MSE loss over 100 epochs for error-correcting encoding using addition

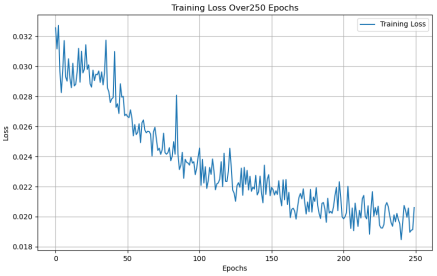

Figure 4.68: Plot for ResNet50 classification training loss with MSE loss over 250 epochs for error-correcting encoding using addition

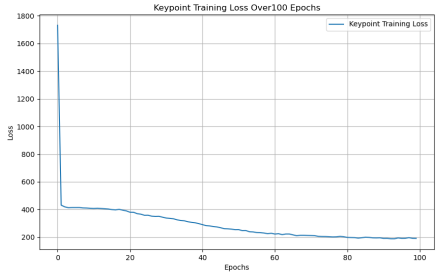

Figure 4.69: Plot for ResNet50 keypoint training loss with MSE loss over 100 epochs for error-correcting encoding using multiplication

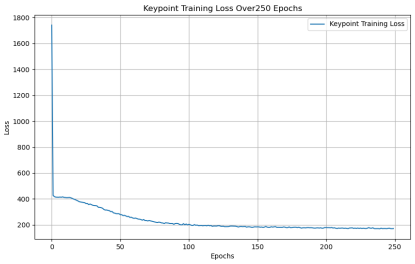

Figure 4.70: Plot for ResNet50 keypoint training loss with MSE loss over 250 epochs for error-correcting encoding using multiplication

Figure 4.71: Plot for ResNet50 classification training loss with MSE loss over 100 epochs for error-correcting encoding using multiplication

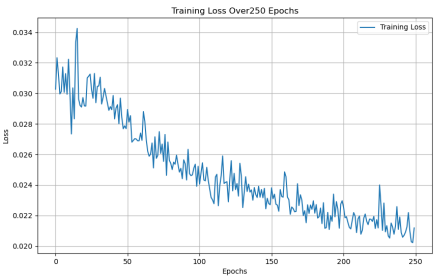

Figure 4.72: Plot for ResNet50 classification training loss with MSE loss over 250 epochs for error-correcting encoding using multiplication

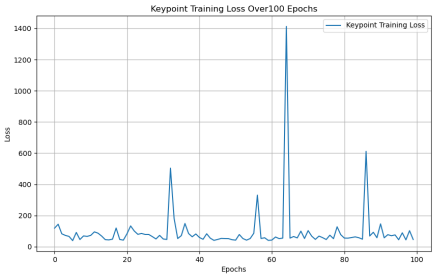

Figure 4.73: Plot for ResNet50 keypoint training loss with Laplace loss over 100 epochs for one-hot encoding using concatenation

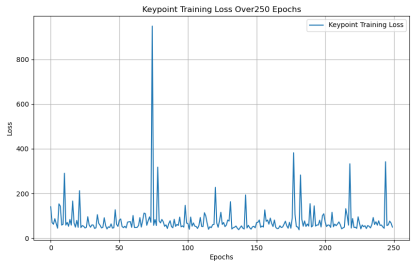

Figure 4.74: Plot for ResNet50 keypoint training loss with Laplace loss over 250 epochs for one-hot encoding using concatenation

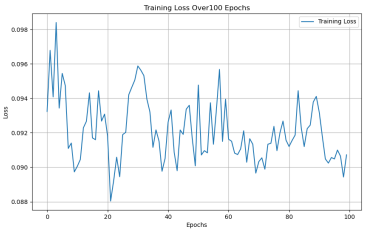

Figure 4.75: Plot for ResNet50 classification training loss with Laplace loss over 100 epochs for one-hot encoding using concatenation

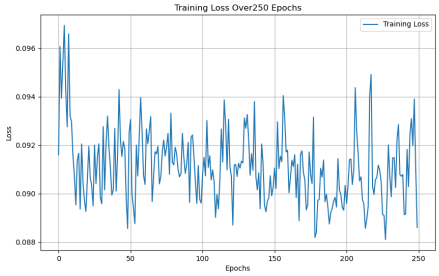

Figure 4.76: Plot for ResNet50 classification training loss with Laplace loss over 250 epochs for one-hot encoding using concatenation

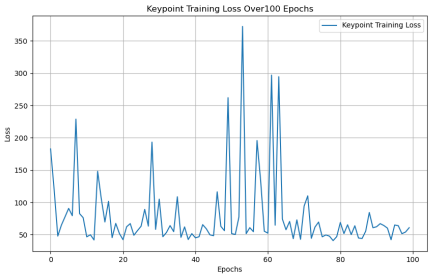

Figure 4.77: Plot for ResNet50 keypoint training loss with Laplace loss over 100 epochs for one-hot encoding using addition

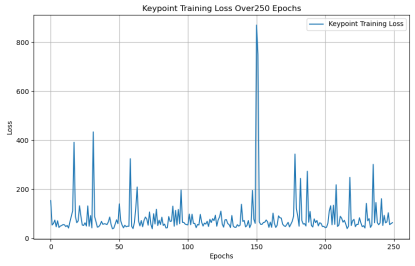

Figure 4.78: Plot for ResNet50 keypoint training loss with Laplace loss over 250 epochs for one-hot encoding using addition

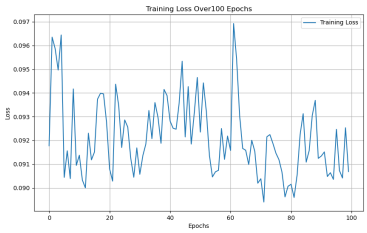

Figure 4.79: Plot for ResNet50 classification training loss with Laplace loss over 100 epochs for one-hot encoding using addition

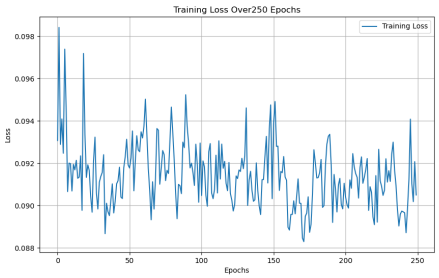

Figure 4.80: Plot for ResNet50 classification training loss with Laplace loss over 250 epochs for one-hot encoding using addition

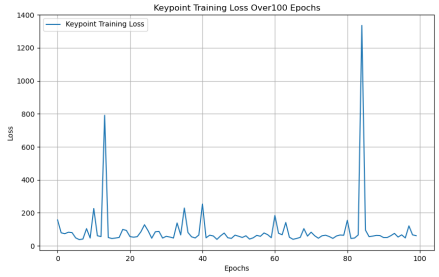

Figure 4.81: Plot for ResNet50 keypoint training loss with Laplace loss over 100 epochs for one-hot encoding using multiplication

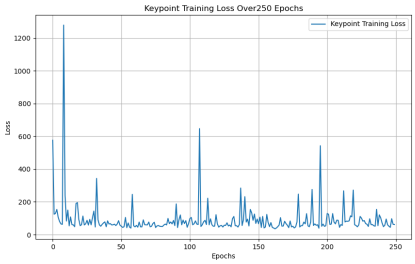

Figure 4.82: Plot for ResNet50 keypoint training loss with Laplace loss over 250 epochs for one-hot encoding using multiplication

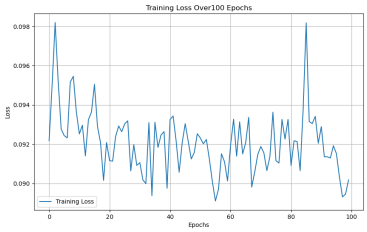

Figure 4.83: Plot for ResNet50 classification training loss with Laplace loss over 100 epochs for one-hot encoding using multiplication

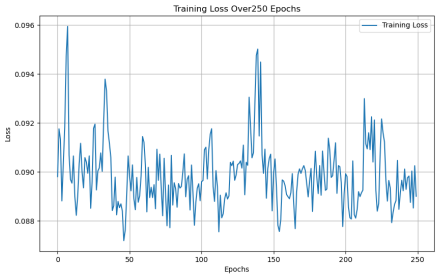

Figure 4.84: Plot for ResNet50 classification training loss with Laplace loss over 250 epochs for one-hot encoding using multiplication

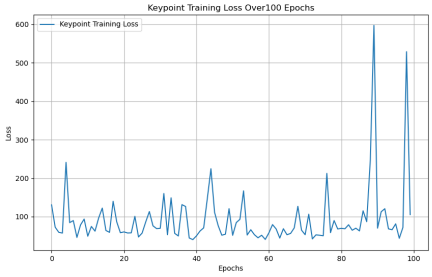

Figure 4.85: Plot for ResNet50 keypoint training loss with Laplace loss over 100 epochs for error-correcting encoding using concatenation

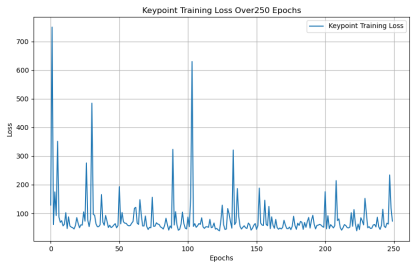

Figure 4.86: Plot for ResNet50 keypoint training loss with Laplace loss over 250 epochs for error-correcting encoding using concatenation

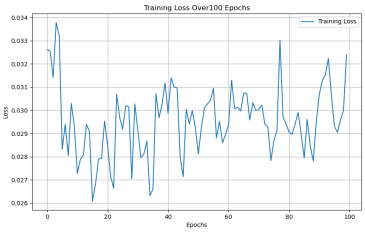

Figure 4.87: Plot for ResNet50 classification training loss with Laplace loss over 100 epochs for error-correcting encoding using concatenation

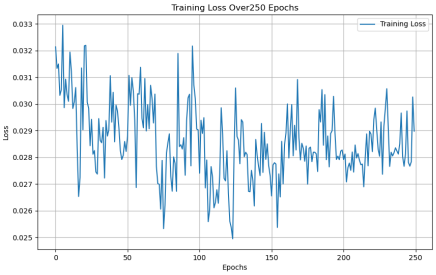

Figure 4.88: Plot for ResNet50 classification training loss with Laplace loss over 250 epochs for error-correcting encoding using concatenation

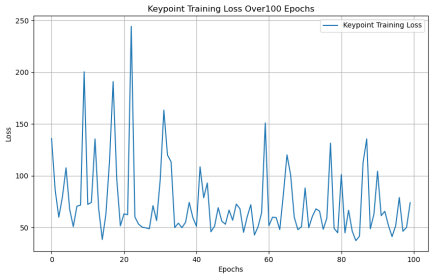

Figure 4.89: Plot for ResNet50 keypoint training loss with Laplace loss over 100 epochs for error-correcting encoding using addition

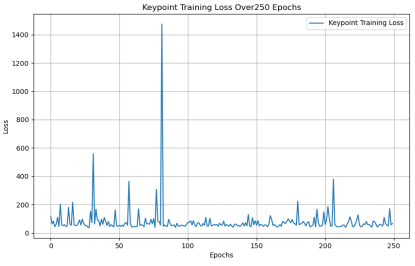

Figure 4.90: Plot for ResNet50 keypoint training loss with Laplace loss over 250 epochs for error-correcting encoding using addition

Figure 4.91: Plot for ResNet50 classification training loss with Laplace loss over 100 epochs for error-correcting encoding using addition

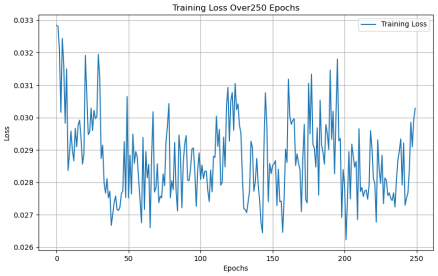

Figure 4.92: Plot for ResNet50 classification training loss with Laplace loss over 250 epochs for error-correcting encoding using addition

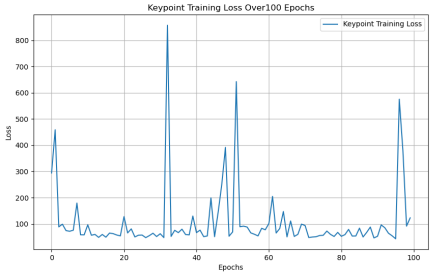

Figure 4.93: Plot for ResNet50 keypoint training loss with Laplace loss over 100 epochs for error-correcting encoding using multiplication

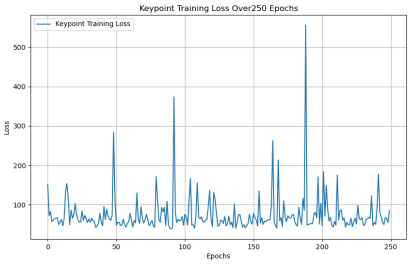

Figure 4.94: Plot for ResNet50 keypoint training loss with Laplace loss over 250 epochs for error-correcting encoding using multiplication

Figure 4.95: Plot for ResNet50 classification training loss with Laplace loss over 100 epochs for error-correcting encoding using multiplication

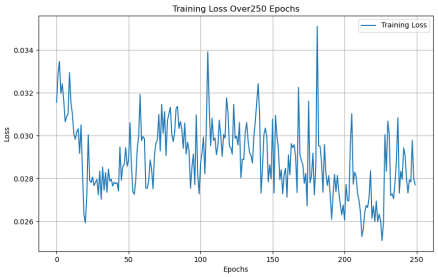

Figure 4.96: Plot for ResNet50 classification training loss with Laplace loss over 250 epochs for error-correcting encoding using multiplication

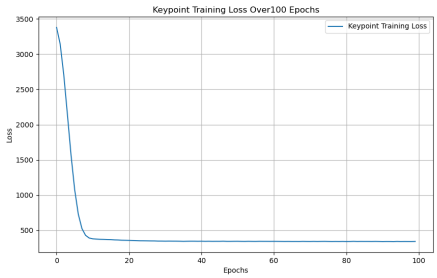

Figure 4.97: Plot for SSD keypoint training loss with MSE loss over 100 epochs for one-hot encoding using concatenation

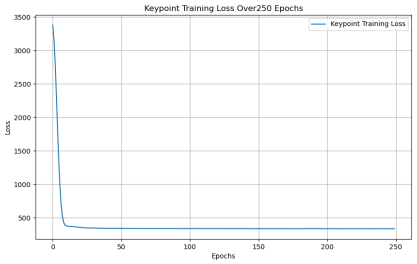

Figure 4.98: Plot for SSD keypoint training loss with MSE loss over 250 epochs for one-hot encoding using concatenation

SSD Network Plots

Using SSD network, for one-hot encoding with MSE loss, the keypoint loss over both the epochs are almost similar for all three fusion methods (concatenation, addition and multiplication) with loss decreasing steadily with very minute fluctuations. After a certain number of epoch values the loss remains almost constant with tendency of no further learning similar to what was observed in ResNet18 network. While the classification loss with Cross-Entropy Loss decreases with fluctuations while training over both the epochs and less fluctuations when compared with ResNet18 network.

Similarly, for error-correcting encoding with MSE loss, the keypoints loss over both the epochs are similar for all three fusion methods (concatenation, addition and multiplication) with loss decreasing to a certain value and then slowing down for some iterations until a stable point i.e around 50-60 epoch values and then the loss curve remains stable with minute fluctuations similar to what was observed in ResNet18 network. While the classification loss with Cross-Entropy Loss decreases with way less fluctuations when compared with one-hot encoding while training over both the epochs. Classification loss for this network is similar to what was observed in ResNet18 network.

For one-hot encoding with Laplace loss, the keypoint loss over both the epochs are almost similar for all three fusion methods (concatenation, addition and multiplication) with loss having fluctuations all during the training and not attaining a stable curve. While the classification loss with Cross-Entropy Loss also have pass through with heavy fluctuations while training and does not attain any stable curve over both the epochs. The case is very similar for error-correcting encoding with Laplace loss for both keypoint and classification. In this case, both the keypoint loss and classification loss trend is similar to what was observed in ResNet18 network.

The plots for these training of dataset for one-hot and error-encoding using both MSE loss and Laplace loss for both keypoint loss and classification loss in SSD network are shown from figures 4.97 to 4.144

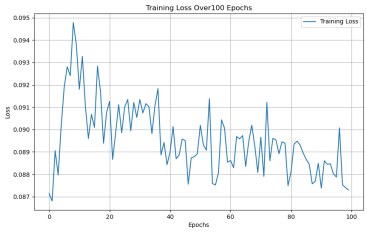

Figure 4.99: Plot for SSD classification training loss with MSE loss over 100 epochs for one-hot encoding using concatenation

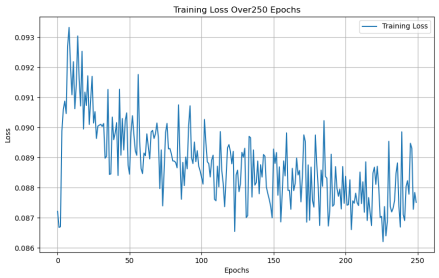

Figure 4.100: Plot for SSD classification training loss with MSE loss over 250 epochs for one-hot encoding using concatenation

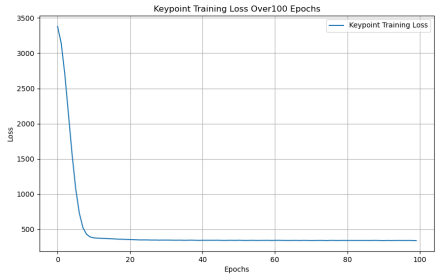

Figure 4.101: Plot for SSD keypoint training loss with MSE loss over 100 epochs for one-hot encoding using addition

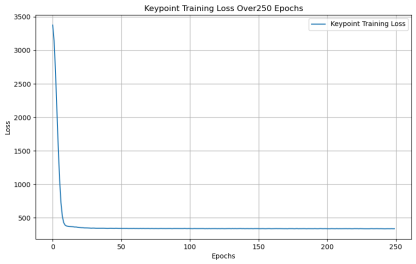

Figure 4.102: Plot for SSD keypoint training loss with MSE loss over 250 epochs for one-hot encoding using addition

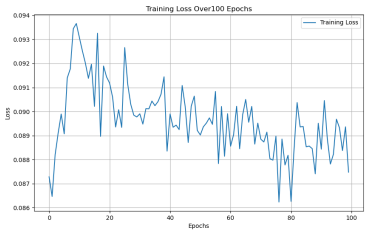

Figure 4.103: Plot for SSD classification training loss with MSE loss over 100 epochs for one-hot encoding using addition

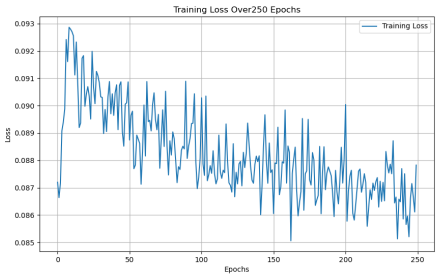

Figure 4.104: Plot for SSD classification training loss with MSE loss over 250 epochs for one-hot encoding using addition

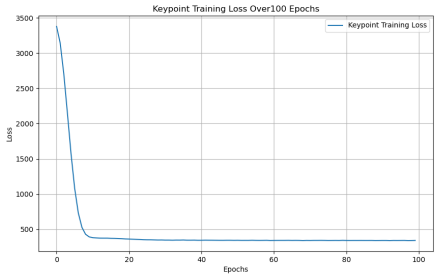

Figure 4.105: Plot for SSD keypoint training loss with MSE loss over 100 epochs for one-hot encoding using multiplication

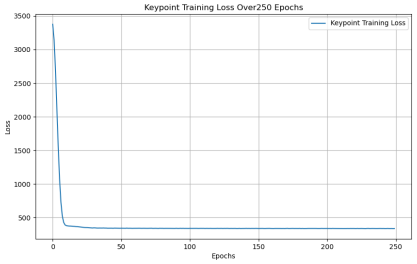

Figure 4.106: Plot for SSD keypoint training loss with MSE loss over 250 epochs for one-hot encoding using multiplication

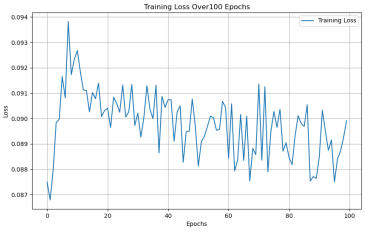

Figure 4.107: Plot for SSD classification training loss with MSE loss over 100 epochs for one-hot encoding using multiplication

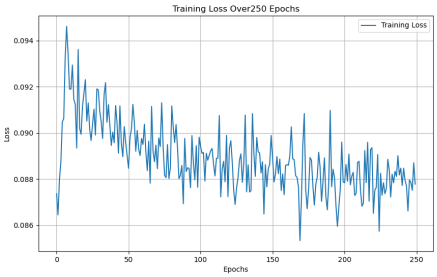

Figure 4.108: Plot for SSD classification training loss with MSE loss over 250 epochs for one-hot encoding using multiplication

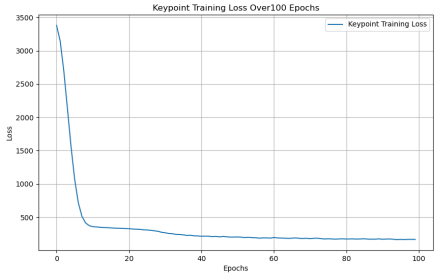

Figure 4.109: Plot for SSD keypoint training loss with MSE loss over 100 epochs for error-correcting encoding using concatenation

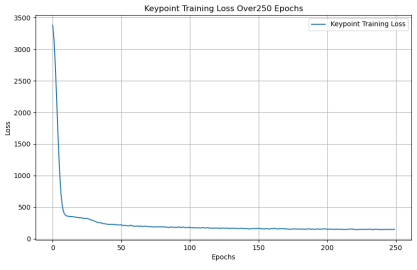

Figure 4.110: Plot for SSD keypoint training loss with MSE loss over 250 epochs for error-correcting encoding using concatenation

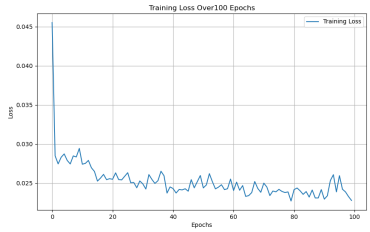

Figure 4.111: Plot for SSD classification training loss with MSE loss over 100 epochs for error-correcting encoding using concatenation

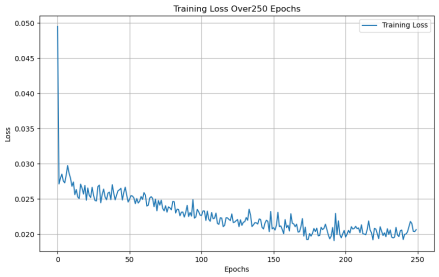

Figure 4.112: Plot for SSD classification training loss with MSE loss over 250 epochs for error-correcting encoding using concatenation

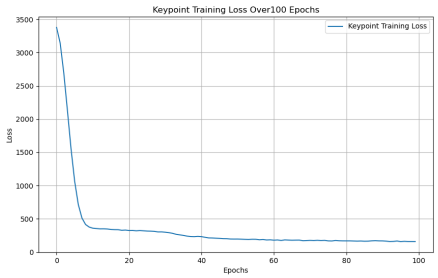

Figure 4.113: Plot for SSD keypoint training loss with MSE loss over 100 epochs for error-correcting encoding using addition

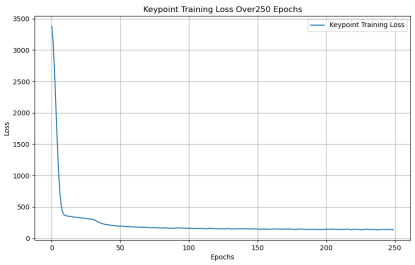

Figure 4.114: Plot for SSD keypoint training loss with MSE loss over 250 epochs for error-correcting encoding using addition

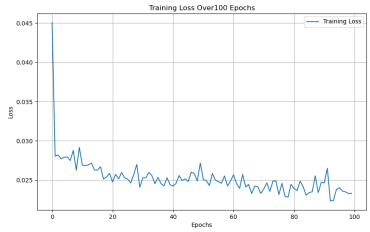

Figure 4.115: Plot for SSD classification training loss with MSE loss over 100 epochs for error-correcting encoding using addition

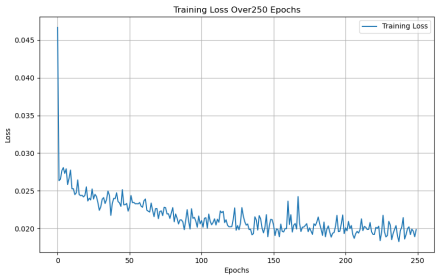

Figure 4.116: Plot for SSD classification training loss with MSE loss over 250 epochs for error-correcting encoding using addition

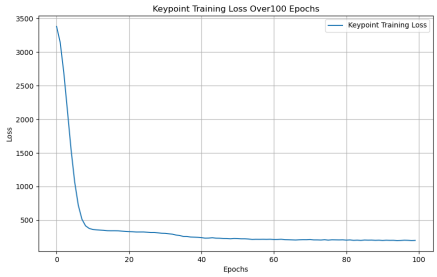

Figure 4.117: Plot for SSD keypoint training loss with MSE loss over 100 epochs for error-correcting encoding using multiplication

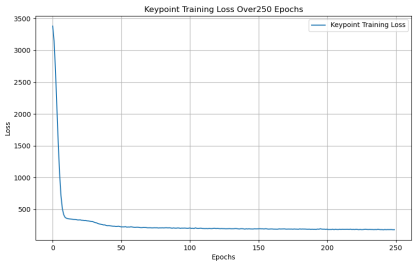

Figure 4.118: Plot for SSD keypoint training loss with MSE loss over 250 epochs for error-correcting encoding using multiplication

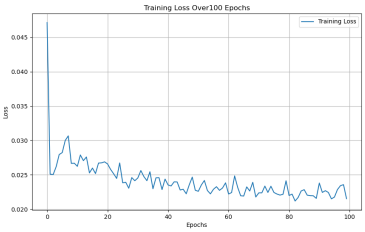

Figure 4.119: Plot for SSD classification training loss with MSE loss over 100 epochs for error-correcting encoding using multiplication

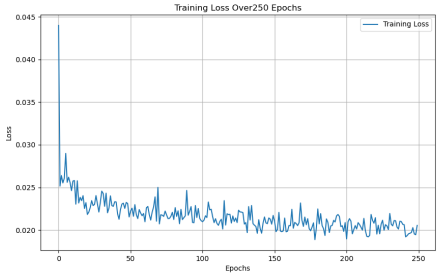

Figure 4.120: Plot for SSD classification training loss with MSE loss over 250 epochs for error-correcting encoding using multiplication

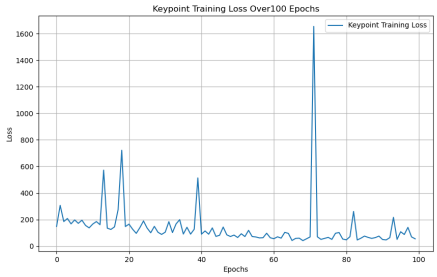

Figure 4.121: Plot for SSD keypoint training loss with Laplace loss over 100 epochs for one-hot encoding using concatenation

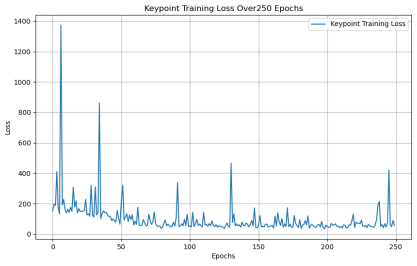

Figure 4.122: Plot for SSD keypoint training loss with Laplace loss over 250 epochs for one-hot encoding using concatenation

Figure 4.123: Plot for SSD classification training loss with Laplace loss over 100 epochs for one-hot encoding using concatenation

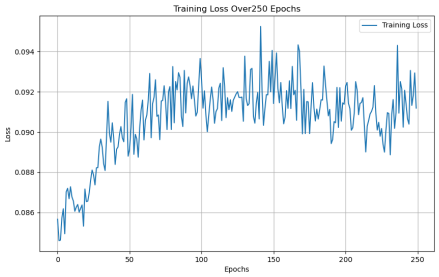

Figure 4.124: Plot for SSD classification training loss with Laplace loss over 250 epochs for one-hot encoding using concatenation

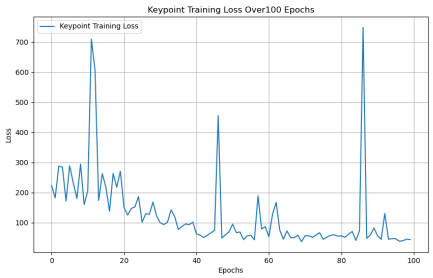

Figure 4.125: Plot for SSD keypoint training loss with Laplace loss over 100 epochs for one-hot encoding using addition

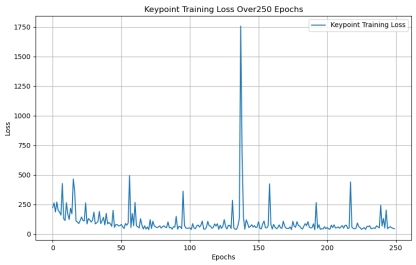

Figure 4.126: Plot for SSD keypoint training loss with Laplace loss over 250 epochs for one-hot encoding using addition

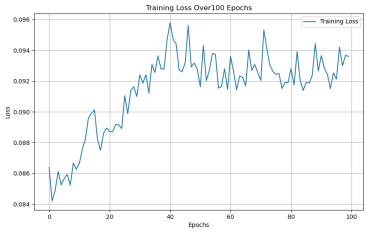

Figure 4.127: Plot for SSD classification training loss with Laplace loss over 100 epochs for one-hot encoding using addition

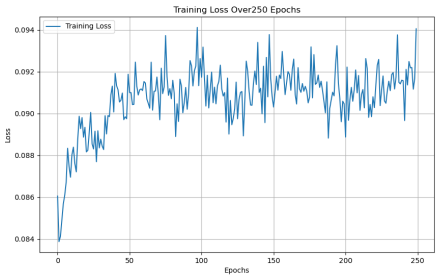

Figure 4.128: Plot for SSD classification training loss with Laplace loss over 250 epochs for one-hot encoding using addition

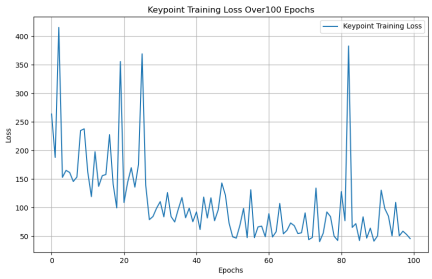

Figure 4.129: Plot for SSD keypoint training loss with Laplace loss over 100 epochs for one-hot encoding using multiplication

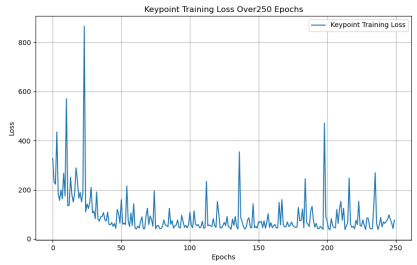

Figure 4.130: Plot for SSD keypoint training loss with Laplace loss over 250 epochs for one-hot encoding using multiplication

Figure 4.131: Plot for SSD classification training loss with Laplace loss over 100 epochs for one-hot encoding using multiplication

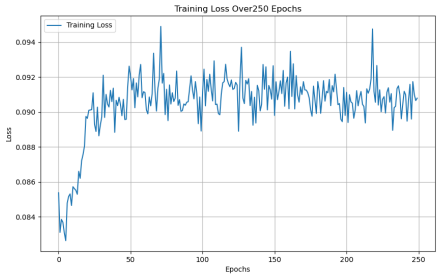

Figure 4.132: Plot for SSD classification training loss with Laplace loss over 250 epochs for one-hot encoding using multiplication

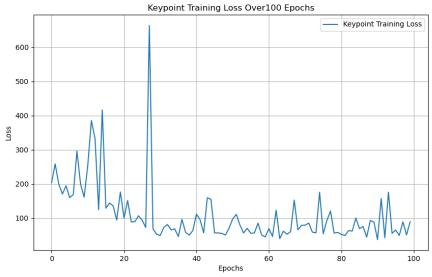

Figure 4.133: Plot for SSD keypoint training loss with Laplace loss over 100 epochs for error-correcting encoding using concatenation

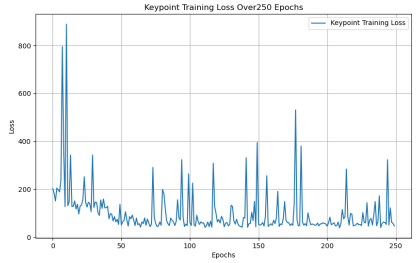

Figure 4.134: Plot for SSD keypoint training loss with Laplace loss over 250 epochs for error-correcting encoding using concatenation

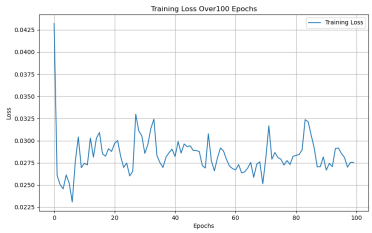

Figure 4.135: Plot for SSD classification training loss with Laplace loss over 100 epochs for error-correcting encoding using concatenation

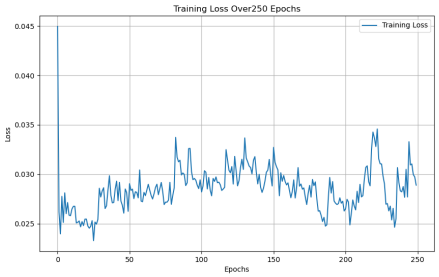

Figure 4.136: Plot for SSD classification training loss with Laplace loss over 250 epochs for error-correcting encoding using concatenation

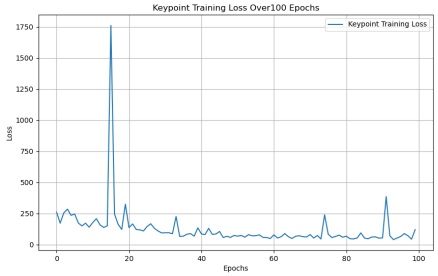

Figure 4.137: Plot for SSD keypoint training loss with Laplace loss over 100 epochs for error-correcting encoding using addition

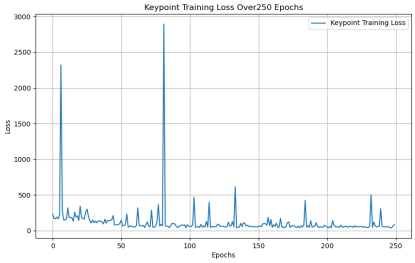

Figure 4.138: Plot for SSD keypoint training loss with Laplace loss over 250 epochs for error-correcting encoding using addition

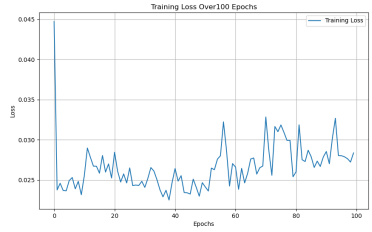

Figure 4.139: Plot for SSD classification training loss with Laplace loss over 100 epochs for error-correcting encoding using addition

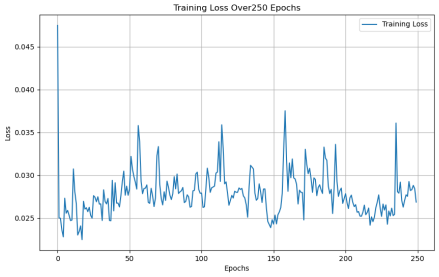

Figure 4.140: Plot for SSD classification training loss with Laplace loss over 250 epochs for error-correcting encoding using addition

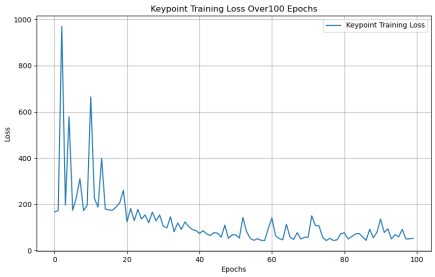

Figure 4.141: Plot for SSD keypoint training loss with Laplace loss over 100 epochs for error-correcting encoding using multiplication

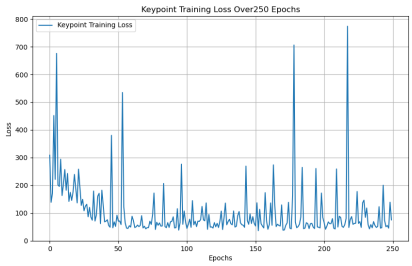

Figure 4.142: Plot for SSD keypoint training loss with Laplace loss over 250 epochs for error-correcting encoding using multiplication

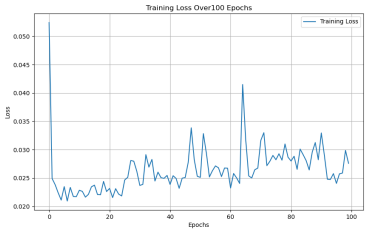

Figure 4.143: Plot for SSD classification training loss with Laplace loss over 100 epochs for error-correcting encoding using multiplication

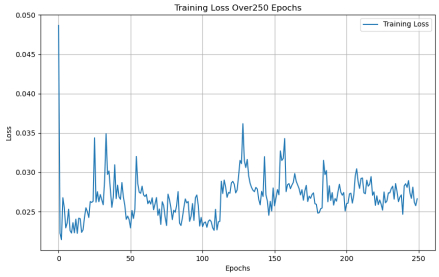

Figure 4.144: Plot for SSD classification training loss with Laplace loss over 250 epochs for error-correcting encoding using multiplication

4.3.2 Tablular Observations

The evaluation of the the studied conditional object detection on the RoboCup@Work dataset is done using metrics like speed(ms), accuracy(Intersection over Union(IoU)) mean Average Precision (mAP) to assess the performance of the trained models and the depiction of this metrics in tabulated as shown in 3.2.

The performance of the Resnet50, ResNet18 architectures using Mean Squared Error (MSE) loss and Laplace is assessed based on different encoding methods (One-hot and Error-correcting) and fusion strategies (Concatenation, Addition, and Multiplication). Key metrics, including speed (in milliseconds), accuracy (measured by Intersection over Union, IoU), and Mean Average Precision (mAP), are provided for both 100 and 250 epochs, allowing for a detailed comparison of how each configuration influences the model's efficiency and effectiveness. These performance metrics are tabulated for ResNet50 using MSE regression loss in table 4.1, for ResNet50 using Laplace regression loss in table 4.2, for ResNet50 using MSE regression loss in table 4.3 and for ResNet50 using Laplace regression loss in table 4.4.

From the table 4.1, below observations are made

- **Speed (ms):** The models perform consistently in terms of speed, with speeds ranging from 21.22 ms to 21.87 ms at 100 epochs, and 21.38 ms to 21.87 ms at 250 epochs.
- **Accuracy (IoU):**
  - The highest accuracy at 250 epochs (0.0072) is achieved with the Error-correcting encoding and Multiplication fusion.
  - The lowest accuracy (0.0054) is observed with Error-correcting encoding and Addition fusion.
- **mean Average Precision (mAP):**
  - The best mAP at 250 epochs (0.0044) is seen with Error-correcting encoding and Concat fusion.
  - The lowest mAP (0.0026) corresponds to Error-correcting encoding with Multiplication fusion.

From the table 4.2, below observations are made

- **Speed (ms):** The speed here is generally higher compared to the MSE loss table, with times ranging from 22.11 ms to 23.23 ms at 100 epochs and 22.55 ms to 23.47 ms at 250 epochs.
- **Accuracy (IoU):**
  - The best accuracy at 250 epochs (0.006) is obtained with Error-correcting encoding and Addition fusion.
  - The lowest accuracy (0.0043) is found with Error-correcting encoding and Multiplication fusion.
- **mean Average Precision (mAP):**

  - The highest mAP (0.0089) is with Error-correcting encoding and Concatenation fusion at 250 epochs.
  - The lowest mAP (0.0014) is with One-hot encoding and Concatenation fusion.

From the table 4.3, below observations are made

- **Speed (ms):** The speed in this case is faster compared to Resnet50, ranging from 7.2 ms to 8.43 ms at 100 epochs and 7.26 ms to 8.47 ms at 250 epochs.
- **Accuracy (IoU):**
  - The highest accuracy at 250 epochs (0.0215) is seen with Error-correcting encoding and Concatenation fusion.
  - The lowest accuracy (0.0056) is with One-hot encoding and Addition fusion.
- **mean Average Precision (mAP):**
  - The best mAP at 250 epochs (0.0089) is also with Error-correcting encoding and Concatenation fusion.
  - The lowest mAP (0.002) is observed with One-hot encoding and Addition fusion.

From the table 4.4, below observations are made

- **Speed (ms):** The speeds are slightly higher than with MSE loss for Resnet18, ranging from 8.17 ms to 8.82 ms at 100 epochs and 7.98 ms to 8.61 ms at 250 epochs.
- **Accuracy (IoU):**
  - The highest accuracy at 250 epochs (0.0215) is achieved with Error-correcting encoding and Concatenation fusion.
  - The lowest accuracy (0.0059) is with Error-correcting encoding and Multiplication fusion.
- **mean Average Precision (mAP):**
  - The highest mAP (0.0118) is with One-hot encoding and Concatenation fusion at 250 epochs.
  - The lowest mAP (0.0053) corresponds to Error-correcting encoding with Multiplication fusion.

Comparison Between Tabulated Data

To evaluate the performance we further compare the the implementation of conditional object detection in these different networks for speed, accuracy and mean average precision values, then followed by the overall observations of encoding methods, fusion approach, loss functions and model network.

- **Speed (ms):**

  - **ResNet50:** Consistently slower than Resnet18, as expected due to its larger architecture. Speeds are similar between MSE loss and Laplace loss, with Laplace loss being slightly slower.
  - **ResNt18:** Much faster than Resnet50. Again, Laplace loss results in slightly slower speeds compared to MSE loss.

- **Accuracy (IoU):**
  - **ResNet50:** Generally achieves lower IoU accuracy across the board compared to Resnet18. The highest accuracy for Resnet50 (0.0072 with MSE loss and Multiplication fusion) is still lower than Resnet18's highest accuracy (0.0215 with MSE loss and Concatenation fusion).
  - **ResNt18:** Consistently achieves higher IoU accuracy, particularly with Error-correcting encoding and Concatenation fusion. Laplace loss also shows slightly lower accuracy compared to MSE loss.

- **mean Average Precision (mAP):**
  - **ResNet50:** Exhibits lower mAP scores, particularly with Laplace Loss. The highest mAP (0.0044 with Error-correcting encoding and Concatenation fusion) is modest.
  - **ResNt18:** Shows better mAP scores overall, with the highest mAP (0.0118 with One-hot encoding and Concatenation fusion) seen with Laplace Loss. MSE Loss also performs well, but with slightly lower scores.

- **Encoding Methods:**
  - Error-correcting encoding generally outperforms One-hot encoding in terms of IoU accuracy and mAP, particularly in Resnet18.

- **Fusion Approach:**
  - Concatenation and Multiplication fusion methods generally yield better accuracy and mAP compared to Addition, though this varies by architecture and loss function.

- **Loss Functions:**
  - MSE loss tends to provide slightly better accuracy and mAP compared to Laplace loss, particularly in the Resnet18 model.

- **Model Network:**
  - Resnet18, being a smaller model, performs faster and generally better in accuracy and mAP than Resnet50 across all configurations.

In general, Resnet18 appears to provide the most optimal equilibrium between speed, accuracy (IoU), and mAP when combined with Concatenation fusion and MSE Loss and Error-correcting encoding. In contrast, Resnet50 exhibits slower performance and generally lower accuracy and mAP scores, rendering

it less appropriate for circumstances in which speed and accuracy are critical. The selection of a loss function and fusion strategy is also a significant factor, with MSE Loss and Concatenation fusion typically yielding superior outcomes.

Architecture	Detection	Encoding	Fusion	Speed(ms)		Accuracy(IoU)		mAP	
				100 Epochs	250 Epochs	100 Epochs	250 Epochs	100 Epochs	250 Epochs
Resnet50	conditional	One-hot	Concatenation	21.22	21.39	0.0078	0.0075	0.0029	0.0028
			Addition	21.49	21.56	0.007	0.0072	0.0025	0.0027
			Multiplication	21.48	21.45	0.0065	0.0069	0.0024	0.0028
		Error-correcting	Concatenation	21.82	21.87	0.0111	0.007	0	0.0044
			Addition	21.31	21.55	0.007	0.0054	0.0037	0.0028
			Multiplication	21.56	21.38	0.0072	0.0072	0.0031	0.0026

Table 4.1: Performance comparison of ResNet50 network with different encoding and fusion methods using MSE regression loss function

Architecture	Detection	Encoding	Fusion	Speed(ms)		Accuracy(IoU)		mAP	
				100 Epo	250 Epo	100 Epo	250 Epo	100 Epo	250 Epo
Resnet50	conditional	One-hot	Concatenation	22.74	22.55	0.0049	0.005	0.0016	0.0014
			Addition	22.11	23.72	0.0052	0.0052	0.0015	0.0016
			Multiplication	23.04	22.78	0.0048	0.0051	0.0014	0.0015
		Error-correcting	Concatenation	22.76	21.82	0.0166	0.0087	0.0089	0.0044
			Addition	23.18	23.15	0.0059	0.0046	0.003	0.0022
			Multiplication	23.09	23.47	0.0047	0.0043	0.0018	0.0015

Table 4.2: Performance comparison of ResNet50 network with different encoding and fusion methods using Laplace regression loss function

Architecture	Detection	Encoding	Fusion	Speed(ms)		Accuracy(IoU)		mAP	
				100 Epo	250 Epo	100 Epo	250 Epo	100 Epo	250 Epo
Resnet18	conditional	One-hot	Concatenation	7.27	7.52	0.0062	0.0069	0.0022	0.0025
			Addition	7.43	7.58	0.0063	0.0056	0.0022	0.002
			Multiplication	7.55	7.94	0.0061	0.0056	0.0024	0.0022
		Error-correcting	Concatenation	8.43	8.27	0.024	0.0215	0.0089	0.0089
			Addition	7.2	7.26	0.0078	0.0062	0.003	0.0022
			Multiplication	7.23	7.46	0.0067	0.0059	0.0022	0.0019

Table 4.3: Performance comparison of ResNet18 network with different encoding and fusion methods using MSE regression loss function

Architecture	Detection	Encoding	Fusion	Speed(ms)		Accuracy(IoU)		mAP	
				100 Epo	250 Epo	100 Epo	250 Epo	100 Epo	250 Epo
Resnet18	conditional	One-hot	Concatenation	8.17	7.98	0.0133	0.0118	0.0044	0.0039
			Addition	8.1	7.71	0.0106	0.0098	0.0035	0.0031
			Multiplication	8.82	8.44	0.0103	0.0095	0.0028	0.0026
		Error-correcting	Concatenation	8.43	8.27	0.024	0.0215	0.0089	0.0089
			Addition	8.49	8.61	0.0198	0.0174	0.0074	0.0056
			Multiplication	8.26	7.94	0.0159	0.0149	0.0053	0.0052

Table 4.4: Performance comparison of ResNet18 network with different encoding and fusion methods using Laplace regression loss function

5

Conclusions

In this research and development of a Multi-Input Conditional Object Detection framework marked a significant advancement in object detection methodologies. The key focus of this research was to improve object detection in specific, targeted scenarios by utilizing conditional inputs such as support images or encoded labels, which differs from traditional methods that detect all objects within a scene. This approach was demonstrated to be particularly effective in industrial applications, such as those seen in the RoboCup@Work competition, where the need to focus on a single object at a time is crucial for task efficiency and accuracy.

The research successfully integrated advanced image processing techniques with deep learning architectures, including ResNet50, ResNet18, and SSD, combined with various data fusion strategies. Through extensive experimentation and evaluation, the study highlighted the effectiveness of these approaches in improving both the computational efficiency and detection accuracy of object detection models. The results showed that conditional object detection could not significantly beat traditional methods with the current approach. The network needed more work and the fused inputs of images and encoded labels needed to be shaped better, especially when there were a lot of things in the background and the sizes of the objects were changing.

This research not only contributed to the development of a specialized dataset for the RoboCup@Work competition but also offered insights into the application of conditional object detection across various real-world scenarios. The findings underline the importance of carefully selecting encoding techniques, fusion strategies, and loss functions to optimize model performance, particularly in resource-constrained environments where speed and accuracy are critical.

5.1 Contributions

In this research, a novel Multi-Input Conditional Object Detection framework was introduced, significantly advancing the current state of object detection methodologies. The primary contribution lies in the development and implementation of a framework that enhances object detection by focusing on specific target objects, utilizing conditional inputs such as support images or encoded labels. This approach differs from traditional object detection methods, which process all objects in a scene, making it particularly useful in industrial applications such as those seen in the RoboCup@Work competition.

Moreover, the research contributed to the creation of a specialized dataset tailored for the RoboCup@Work competition, featuring 18 distinct object classes. This dataset, along with advanced image processing techniques, enabled the training of deep learning models, including ResNet50, ResNet18, and SSD, integrated with various data fusion techniques. These contributions have not only improved the accuracy and efficiency of object detection models but have also opened avenues for further research in applying conditional object detection to a broader range of real-world scenarios.

5.2 Lessons learned

Several key lessons were learned throughout the course of this research. Firstly, the importance of encoding techniques was underscored, with error-correcting encoding generally outperforming one-hot encoding, particularly in the ResNet18 model. This finding emphasizes the role of encoding in enhancing the accuracy and robustness of object detection models.

Secondly, the study highlighted the varying effectiveness of different fusion strategies. Concat and multiplication fusion methods generally yielded better accuracy and mean Average Precision (mAP) compared to addition, although the optimal strategy varied depending on the model architecture and loss function used. This indicates the need for careful selection and tuning of fusion strategies to maximize model performance.

Lastly, the research demonstrated that the choice of loss function could significantly impact model performance. MSE Loss tended to offer slightly better accuracy and mAP compared to Laplace Loss, particularly in the smaller ResNet18 model. This finding suggests that while larger models like ResNet50 might benefit from different loss functions, smaller models might require more traditional approaches to achieve optimal results.

5.3 Future work

The findings from this research pave the way for several future research directions. One promising area for future work is the refinement of the conditional object detection model. Further exploration of additional fusion techniques could lead to even greater improvements in detection accuracy and efficiency. Additionally, applying the conditional object detection framework to a wider variety of real-world scenarios could enhance its adaptability and impact.

Another potential direction is the exploration of other deep learning architectures beyond ResNet and SSD, which may offer additional improvements in speed and accuracy. Investigating the integration of the conditional object detection framework with real-time processing systems, such as those used in autonomous vehicles and robotics, could also be a valuable extension of this work.

Finally, there is potential for enhancing the dataset used in this research by including more object classes and more varied scenes. This could further test the robustness and generalization capabilities of the conditional object detection framework, ensuring its effectiveness across an even broader range of applications.

A

Design Details

A comprehensive description of the specific design choices made during the development of Multi-Input Conditional Object Detection framework are mentioned below. Additionally, AI assistant tools such as ChatGPT-4, QuillBot and Perplexity were used to gather some information about the recent research papers and were cross-examined to check their validity.

- **Model Network:**
  - Detailed descriptions of the architectures used (e.g., ResNet50, ResNet18, SSD).
  - Specific modifications or customizations made to these architectures to support conditional object detection.
  - The reasoning behind choosing these particular models and how they contribute to the overall framework.
- **Fusion Techniques:**
  - Detailed explanations of the fusion methods implemented (Concatenation, Addition, Multiplication).
  - Visual representations (diagrams or flowcharts) of how these fusion techniques were integrated into the models.
  - Challenges encountered during the implementation of these fusion techniques and how they were addressed.
- **Encoding Strategies:**
  - Detailed information on the encoding techniques used (One-hot, Error-correcting).
  - Explanation of how these encodings were generated and applied within the model.
  - Pre-processing steps or transformations applied to the encoded data before fusion with image data.
- **Data Processing and Preparation:**

- Details on how the dataset was prepared, including any augmentation techniques, normalization processes, or resizing operations.
- How the dataset was divided for training, validation, and testing.
- The steps involved in labeling the dataset and ensuring the quality and consistency of the labels.

- **Training Methodology:**
  - Explanation of the training pipeline, including batch sizes, number of epochs, and other relevant parameters.
  - Strategies used for loss calculation, optimization, and model evaluation during training.
  - Any specific techniques used to prevent overfitting, such as dropout, regularization, or early stopping.

B

Parameters

This appendix lists and explain all the parameters used during the experimentation phase of the research.

- **Hyperparameters:**
  - Detailed lists of all hyperparameters used for each model, including learning rates, batch sizes, number of epochs, and optimizer settings (e.g., Adam, SGD).
  - Rationale for choosing specific values for these hyperparameters and any tuning strategies employed.
- **Loss Functions:**
  - Detailed descriptions of the loss functions used (e.g., MSE Loss, Laplace Loss).
  - Parameters specific to the loss functions, such as weight decay or learning rate schedules.
  - Justifications for the choice of loss functions and how they were implemented in the models.
- **Fusion Parameters:**
  - Any specific parameters related to the fusion methods, such as the axis of concatenation or scaling factors for addition or multiplication.
  - How these parameters were optimized or adjusted during experimentation.
- **Dataset Parameters:**
  - Information on the input image sizes, data augmentation settings, and any other dataset-specific parameters.
  - Details on how the dataset was split into training, validation, and test sets, including the percentage split and selection criteria.
- **Evaluation Metrics:**
  - Explanation of the metrics used to evaluate the models (e.g., Accuracy (IoU), mean Average Precision (mAP), speed in ms).
  - Any threshold values or specific criteria used in the evaluation process.

References

[1] S. Lang, F. Ventola, and K. Kersting, "Dafne: A one-stage anchor-free approach for oriented object detection," *arXiv*, 2022. [Online]. Available: https://arxiv.org/abs/2109.06148

[2] Y. He, C. Zhu, J. Wang, M. Savvides, and X. Zhang, "Bounding box regression with uncertainty for accurate object detection," *arXiv*, 2019. [Online]. Available: https://arxiv.org/abs/1809.08545

[3] Z. Liu, Z. Wu, and R. Tóth, "Smoke: Single-stage monocular 3d object detection via keypoint estimation," *arXiv*, 2020. [Online]. Available: https://arxiv.org/abs/2002.10111

[4] Y. Lin, J. Tremblay, S. Tyree, P. A. Vela, and S. Birchfield, "Single-stage keypoint-based category-level object pose estimation from an rgb image," *arXiv*, 2022. [Online]. Available: https://arxiv.org/abs/2109.06161

[5] R. Federation. Robocup@work. [Online]. Available: https://atwork.robocup.org/

[6] K. Fu, T. Zhang, Y. Zhang, and X. Sun, "Oscd: A one-shot conditional object detection framework," *Neurocomputing*, vol. 425, pp. 243–255, 2020. [Online]. Available: https://api.semanticscholar.org/CorpusID:219004774

[7] J. Redmon and A. Farhadi, "Yolo9000: Better, faster, stronger," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 6517–6525.

[8] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, *SSD: Single Shot MultiBox Detector*. Springer International Publishing, 2016, p. 21–37. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-46448-0_2

[9] K. Chen, J. Li, W. Lin, J. See, J. Wang, L. Duan, Z. Chen, C. He, and J. Zou, "Towards accurate one-stage object detection with ap-loss," 2020. [Online]. Available: https://arxiv.org/abs/1904.06373

[10] Q. Fan, W. Zhuo, C.-K. Tang, and Y.-W. Tai, "Few-shot object detection with attention-rpn and multi-relation detector," 06 2020, pp. 4012–4021.

[11] B. Kang, Z. Liu, X. Wang, F. Yu, J. Feng, and T. Darrell, "Few-shot object detection via feature reweighting," *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 8419–8428, 2018. [Online]. Available: https://api.semanticscholar.org/CorpusID:54459557

[12] X. Hu, X. Xu, Y. Xiao, H. Chen, S. He, J. Qin, and P.-A. Heng, "Sinet: A scale-insensitive convolutional neural network for fast vehicle detection," *IEEE Transactions on Intelligent Transportation Systems*, vol. PP, 04 2018.

[13] T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie, "Feature pyramid networks for object detection," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 936–944.

[14] X. Zhou, D. Wang, and P. Krähenbühl, "Objects as points," 2019. [Online]. Available: https://arxiv.org/abs/1904.07850

[15] S. Zhang, L. Wen, X. Bian, Z. Lei, and S. Z. Li, "Single-shot refinement neural network for object detection," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018, pp. 4203–4212.

[16] L. Huang, Y. Yang, Y. Deng, and Y. Yu, "Densebox: Unifying landmark localization with end to end object detection," 09 2015.

[17] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *2014 IEEE Conference on Computer Vision and Pattern Recognition*, 2014, pp. 580–587.

[18] R. Girshick, "Fast r-cnn," in *2015 IEEE International Conference on Computer Vision (ICCV)*, 2015, pp. 1440–1448.

[19] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778.

[20] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," 2016. [Online]. Available: https://arxiv.org/abs/1506.01497

[21] J. Huang, V. Rathod, C. Sun, M. Zhu, A. Korattikara, A. Fathi, I. Fischer, Z. Wojna, Y. Song, S. Guadarrama, and K. Murphy, "Speed/accuracy trade-offs for modern convolutional object detectors," 2017. [Online]. Available: https://arxiv.org/abs/1611.10012

[22] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 779–788.

[23] C.-Y. Fu, W. Liu, A. Ranga, A. Tyagi, and A. C. Berg, "Dssd : Deconvolutional single shot detector," 2017. [Online]. Available: https://arxiv.org/abs/1701.06659

[24] N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov, and S. Zagoruyko, "End-to-end object detection with transformers," 2020. [Online]. Available: https://arxiv.org/abs/2005.12872

[25] P. Sun, R. Zhang, Y. Jiang, T. Kong, C. Xu, W. Zhan, M. Tomizuka, L. Li, Z. Yuan, C. Wang, and P. Luo, "Sparse r-cnn: End-to-end object detection with learnable proposals," 2021. [Online]. Available: https://arxiv.org/abs/2011.12450

[26] K. Duan, S. Bai, L. Xie, H. Qi, Q. Huang, and Q. Tian, "Centernet: Keypoint triplets for object detection," in *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, 2019, pp. 6568–6577.

[27] V. Boykis, "What are embeddings?" Jun. 2023. [Online]. Available: https://github.com/veekaybee/what_are_embeddings

[28] Z. Harris, "Distributional structure," *Word*, vol. 10, no. 2-3, pp. 146–162, 1954. [Online]. Available: https://link.springer.com/chapter/10.1007/978-94-009-8467-7_1

[29] K. S. Jones, "A statistical interpretation of term specificity and its application in retrieval," *J. Documentation*, vol. 60, pp. 493–502, 2021. [Online]. Available: https://api.semanticscholar.org/CorpusID:2996187

[30] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," 2013. [Online]. Available: https://arxiv.org/abs/1301.3781

[31] O. Levy, Y. Goldberg, and I. Dagan, "Improving distributional similarity with lessons learned from word embeddings," *Transactions of the Association for Computational Linguistics*, vol. 3, pp. 211–225, 2015. [Online]. Available: https://aclanthology.org/Q15-1016

[32] J. Pennington, R. Socher, and C. Manning, "Glove: Global vectors for word representation," vol. 14, 01 2014, pp. 1532–1543.

[33] W. Yin, K. Kann, M. Yu, and H. Schütze, "Comparative study of cnn and rnn for natural language processing," 2017. [Online]. Available: https://arxiv.org/abs/1702.01923

[34] R. Johnson and T. Zhang, "Effective use of word order for text categorization with convolutional neural networks," 2015. [Online]. Available: https://arxiv.org/abs/1412.1058

[35] C. Guo and F. Berkhahn, "Entity embeddings of categorical variables," 2016. [Online]. Available: https://arxiv.org/abs/1604.06737

[36] D. Hand and R. Till, "A simple generalisation of the area under the roc curve for multiple class classification problems," *Hand, The*, vol. 45, pp. 171–186, 11 2001.

[37] T. G. Dietterich and G. Bakiri, "Solving multiclass learning problems via error-correcting output codes," 1995. [Online]. Available: https://arxiv.org/abs/cs/9501101

[38] S. Escalera, O. Pujol, and P. Radeva, "Error-correcting ouput codes library," *Journal of Machine Learning Research*, vol. 11, pp. 661–664, 02 2010.

[39] K. Simonyan and A. Zisserman, "Two-stream convolutional networks for action recognition in videos," 2014. [Online]. Available: https://arxiv.org/abs/1406.2199

[40] G. Barnum, S. Talukder, and Y. Yue, "On the benefits of early fusion in multimodal representation learning," *arXiv*, 2020. [Online]. Available: https://arxiv.org/abs/2011.07191

[41] D. Xu, Y. Zhu, C. B. Choy, and L. Fei-Fei, "Scene graph generation by iterative message passing," 2017. [Online]. Available: https://arxiv.org/abs/1701.02426

[42] X. Chen, H. Ma, J. Wan, B. Li, and T. Xia, "Multi-view 3d object detection network for autonomous driving," 2017. [Online]. Available: https://arxiv.org/abs/1611.07759

[43] J. Ku, M. Mozifian, J. Lee, A. Harakeh, and S. Waslander, "Joint 3d proposal generation and object detection from view aggregation," 2018. [Online]. Available: https://arxiv.org/abs/1712.02294

[44] T. Laidlow, J. Czarnowski, and S. Leutenegger, "Deepfusion: Real-time dense 3d reconstruction for monocular slam using single-view depth and gradient predictions," 2022. [Online]. Available: https://arxiv.org/abs/2207.12244

[45] Y. Zhou and O. Tuzel, "Voxelnet: End-to-end learning for point cloud based 3d object detection," 2017. [Online]. Available: https://arxiv.org/abs/1711.06396

[46] Z. Kang, P. Zhang, X. Zhang, J. Sun, and N. Zheng, "Instance-conditional knowledge distillation for object detection," 2022. [Online]. Available: https://arxiv.org/abs/2110.12724

[47] Z. Chen, K. Sun, X. Lin, and R. Ji, "Camodiffusion: Camouflaged object detection via conditional diffusion models," 2023. [Online]. Available: https://arxiv.org/abs/2305.17932

[48] D. Meng, X. Chen, Z. Fan, G. Zeng, H. Li, Y. Yuan, L. Sun, and J. Wang, "Conditional detr for fast training convergence," 2023. [Online]. Available: https://arxiv.org/abs/2108.06152

[49] M. Yamada, "Detect only what you specify : Object detection with linguistic target," Nov. 2022.

[50] Y. Zang, W. Li, K. Zhou, C. Huang, and C. C. Loy, "Open-vocabulary detr with conditional matching," pp. 106–122, Mar. 2022.

[51] R. Jiang, L. Liu, and C. Chen, "Clip-count: Towards text-guided zero-shot object counting," Oct. 2023.

[52] Intel. Intel realsense d435. [Online]. Available: https://www.intelrealsense.com/depth-camera-d435/

[53] W. Wang, "Advanced auto labeling solution with added features," https://github.com/CVHub520/X-AnyLabeling, CVHub, 2023.

[54] MAS-group. b-it-bots@work. [Online]. Available: https://www.h-brs.de/en/a2s/b-it-bots

[55] K. Patel, G. Chenchani, R. Selvaraju, S. Shinde, V. Kalagaturu, V. Mannava, S. Thoduka, and D. Nair, "RoboCup @Work 2023 dataset," 2023. [Online]. Available: https://zenodo.org/records/10003915