2022

ENHANCED DEEP TEMPLATE-BASED OBJECT INSTANCE DETECTION

Travis Frink
University of Rhode Island, tfrink238@gmail.com

ENHANCED DEEP TEMPLATE-BASED OBJECT INSTANCE DETECTION

BY

TRAVIS FRINK

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE

REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE

IN

ELECTRICAL ENGINEERING

UNIVERSITY OF RHODE ISLAND

2022

MASTER OF SCIENCE THESIS

OF

TRAVIS FRINK

APPROVED:

Thesis Committee:

Major Professor Resit Sendag

Manbir Sodhi

Yan Sun

Hui Lin

Brenton DeBoef

DEAN OF THE GRADUATE SCHOOL

UNIVERSITY OF RHODE ISLAND

2022

ABSTRACT

Deep convolutional neural networks (CNNs) have greatly accelerated computer vision tasks such as object detection in recent years. Still, their applicability is limited in some cases due to their need for data. For example, in systems which frequently learn new objects. Deep Template-based Object Instance Detection (DTOID) is a model potentially useful for this task. Common methods require training with many examples labeled by a human to detect new objects. DTOID is unique due to only requiring a single 3D model of an object of interest (target). This unique property makes DTOID especially useful in scenarios where new objects are frequently learned. However, DTOID's object detection performance has a significant room for improvement.

Due to DTOID's potential usefulness but lack of object detection performance this study focuses on improving DTOID. Simple scenes with textureless objects are considered to replicate industrial settings where CAD models are readily available. This study contributes three alterations to DTOID and answers the question of if these alterations can improve DTOID's object (instance) detection performance. The first alteration investigated is the addition of a feature pyramid network (FPN) to DTOID's backbone. In addition, a novel method called viewpoint loss weighting is proposed which provides more importance to target samples with less robustness perturbation. Lastly, a transformer is integrated into DTOID to replace the methods used to aggregate information from input images and target data.

As a result of this investigation it was found that the addition of a FPN into DTOID and the proposed viewpoint loss weighting strategy improve performance. However, the addition of a transformer did not improve performance. The poor performance of the transformer variant of DTOID is investigated, with a possible flaw discussed.

ACKNOWLEDGMENTS

I am grateful to Professor Sendag and Professor Sodhi for providing me the opportunity to do research in this interesting area. They provided me helpful guidance and resources throughout my graduate experience. I would also like to thank Professor Sun and Professor Lin for being on my committee and for Professor Taggart for serving as my defense chair.

TABLE OF CONTENTS

Page

Page

Page

LIST OF FIGURES

Figure **Page**

LIST OF TABLES

CHAPTER 1

Introduction

In the past decade great progress has been made in allowing computers to understand the world through media such as audio and especially images. The progress made in computer-based visual understanding, known as computer vision, has been made largely possible through deep neural networks, especially convolutional neural networks (CNNs) [1]. One area of computer vision accelerated by deep learning is a task known as object detection. Object detection involves the process of both locating and identifying objects of interest in an image. Today deep learning is being increasingly applied in many industries. For example, to detect pedestrians for collision avoidance systems in autonomous cars [2], analyze farmland in agricultural settings [3], and analyzing defects in manufactured components [4].

While CNN based object detection models are highly effective they are limited in their ability to learn new objects to detect. Training a CNN based detector from scratch can require thousands of images at a minimum. As a result, problems which involve constantly adding new objects to classify would become impractical given such assumptions. However, prior knowledge of similar problems can be used to enhance training for new ones. This process is known as transfer-learning and is the primary way many machine learning problems are solved without requiring impractical quantities of data.

The process of training a model for a new task which has already been trained to perform a related task is known as fine-tuning. Few-shot methods specifically focus on the task of training models with minimal amounts of data and often involve fine-tuning. In the context of few-shot methods, images containing the object of

interest are called support images while the image to perform object detection on is called the query image. While fine-tuning methods have been shown to be effective they still have the fundamental limitations of requiring labeled data and training. Training itself can require tens or hundreds of model evaluations as well as updates to model parameters. This increases computation time and may require increased complexity and implementation costs for specialized hardware solutions.

Meta-learning is another method which can be used to perform few-shot object detection. Support images are used as model inputs rather than for training. Features created with support images are then combined with the features of the query image to perform object detection. While computation is still required to process support images it is typically considerably less than that compared to training since only a single evaluation is required for each image example and no weight updates are required. Also, it is common for support branches to have smaller input images, further reducing computational cost.

Zero-shot methods attempt to solve the problem of data collection and labeling completely by using data which is correlated with a desired target object rather than using images of the object itself. Deep Template-based Object Instance Detection (DTOID) is a model which can be considered zero-shot due to the fact that it is designed to use synthetic support images (templates) rather than real ones. By using synthetic images only a single 3D model of the desired target object is required. 3D CAD models are often readily available for components manufactured in industrial settings, a focus area of this study.

Object detection performance or accuracy can be described as how well a given object detector is able to find and locate objects of interest in a given query image. While DTOID provides a significant advantage over trained models and even typical meta-learning few-shot object detectors in terms of data requirements, it is still

highly limited by object detection performance. This study focuses on applying a combination of novel and existing methods to improve the performance of DTOID in terms of object detection performance. 5 chapters provide an overview of deep learning for object detection, related work, methods, experimental results and a conclusion. Ultimately this work contributes three methods of altering DTOID and an experimental investigation demonstrating the ability of those methods to enhance DTOID's performance.

Chapter 2 provides an overview of object detection using deep learning. Multilayer perceptrons (MLPs) are discussed in detail which form a basis for convolutional neural networks (CNNs). Convolutional neural networks are then described, including methods that allow the training of deep CNNs. Methods related to object detection are then discussed, including one-stage detectors, two-stage detectors, and feature pyramid networks. Finally, other deep learning methods applicable to this study are introduced such as image segmentation and transformers.

Chapter 3 focuses on work directly related to the way this study presents DTOID. In particular, methods for few-shot object detection are discussed. Successful fine-tuned and meta-learning methods are described and compared, including methods which are similar to those used in DTOID. Several problems involving data and training requirements are researched. It is found that DTOID requires significantly less effort in terms of collecting data compared to other methods, especially in scenarios where 3D models of objects to recognize are available. In addition, DTOID is reviewed in detail in terms of its architecture and training methodologies. However, the original DTOID work shows object detection performance with a significant room for improvement.

Chapter 4 discusses methods which are broken up into the alterations investigated and data creation. Three alterations to DTOID are proposed and discussed

with a primary research question of if these methods can improve the object (instance) detection performance of DTOID. Object detection performance is measured based on six metrics from the popular object detection benchmark COCO [5]. Feature pyramid networks (FPNs) [6] are a well studied object detection subnetwork which are used to improve performance on small objects. The first alteration integrates a FPN into DTOID. During training DTOID perturbs the view of objects present in template images to increase robustness. However, examples with greater perturbation are given the same value. The second alteration is a novel viewpoint loss weighting strategy which biases training towards template examples which more accurately correspond with the query image. Lastly, transformers [7] are a recently highly investigated deep learning method applied to a variety of domains such as computer vision. The third alteration is the addition of a transformer to replace current methods used to aggregate query and template features. The data sampling and generation process used for training and evaluation is discussed in the second part of chapter 4. Emphasis is put on untextured components in simple scenes similar to industrial settings. The algorithm used to choose objects for training and testing is described, as well as the process used to creating query and template image data.

Finally, in Chapter 5 experimental results are presented and discussed. It is found that the addition of an FPN is able to substantially increase performance. In addition, viewpoint loss weighting appears to provide a modest increase in performance when considering trial averages. Lastly, the transformer model is functional but does not have performance on par with the base model. The cause of failure in the transformer variant of DTOID is investigated, with a possible flaw in it's design discussed. A conclusion is then provided summarizing results and discussing future work.

List of References

[1] Z. Li, W. Yang, S. Peng, and F. Liu, "A survey of convolutional neural networks: Analysis, applications, and prospects," *CoRR*, vol. abs/2004.02806, 2020.

[2] Z. Yang, J. Li, and H. Li, "Real-time pedestrian detection for autonomous driving," in *2018 International Conference on Intelligent Autonomous Systems (ICoIAS)*, 2018, pp. 9–13.

[3] A. Kamilaris and F. X. Prenafeta-Boldú, "Deep learning in agriculture: A survey," *Computers and Electronics in Agriculture*, vol. 147, pp. 70–90, 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0168169917308803

[4] J. Yang, S. Li, Z. Wang, H. Dong, J. Wang, and S. Tang, "Using deep learning to detect defects in manufacturing: A comprehensive survey and current challenges," *Materials*, vol. 13, no. 24, 2020. [Online]. Available: https://www.mdpi.com/1996-1944/13/24/5755

[5] "Coco metrics," https://cocodataset.org, accessed: 2022-06-23.

[6] T. Lin, P. Dollár, R. B. Girshick, K. He, B. Hariharan, and S. J. Belongie, "Feature pyramid networks for object detection," *CoRR*, vol. abs/1612.03144, 2016. [Online]. Available: http://arxiv.org/abs/1612.03144

[7] T. Lin, Y. Wang, X. Liu, and X. Qiu, "A survey of transformers," *CoRR*, vol. abs/2106.04554, 2021. [Online]. Available: https://arxiv.org/abs/2106.04554

CHAPTER 2

Convolutional Neural Networks for Object Detection

2.1 Machine Learning and Neural Networks

Deep learning itself is a sub-sect of machine learning or ML. A common definition used for ML is [1]:

> A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E.

The experience E can being anything from a few measurements to an image or more and is usually the input to a machine learning model. The task T is then the operation which the machine learning model is trained to perform, for example identifying E which is called classification or providing an action based on E such as the direction to move a vehicle. P then provides a objective measure of the quality on T given E. P is often what is used to train a model so that it can learn.

2.1.1 Machine Learning

Machine learning algorithms are characterized by their ability to use experience to improve their performance on a given task. A non-ML algorithm would then be one which does not learn, and has all decision making processes defined by a human designer. ML is often used to deal with problems involving information which is often regarded as being too difficult to reason with by means of human-described decision making processes. ML can be seen as a subsect of artificial intelligence, as ML can be used to reason and interpret information which typically can only be understood by human intelligence directly. For example, deep learning machine learning models have excelled at tasks such as recognizing the type of object that is in an image, or translating text from one language to another.

Supervised machine learning is one of the most common machine learning methods used and researched. Supervised machine learning methods deal with tasks that involve matching a set of inputs $\mathbf{x} \in \mathcal{X}$, called features (also known as covariates or predictors) to a set of outputs $\mathbf{y} \in \mathcal{Y}$, called labels (also known as targets or responses) [2]. Both features and labels are most often represented numerically as most machine learning algorithms are numerical in nature. A set of N input-output pairs $\mathcal{D} = \{(\mathbf{x}_n, \mathbf{y}_n)\}_{n=1}^{N}$ can then be used to provide a machine learning model experience. $\mathcal{D}$ can be used to train the model in which case $\mathcal{D}$ may be called a training dataset. $\mathcal{D}$ may also be used to evaluate the model's performance, in which case $\mathcal{D}$ may be called a testing dataset.

Features can be any data associated with the task at hand. It is common for features to be a vector so $\mathcal{X} = \mathbb{R}^D$, with each element containing a single feature. For example, if a task involves classifying birds then two possible features could be wingspan and height, in which case $\mathcal{X} = \mathbb{R}^2$.

Two common problems which are often solved through supervised learning include classification and regression. In classification the labels $\mathcal{Y} = \{1, 2, ..., C\}$ are a set of natural numbers where each value belongs to a specific type of class. Binary classification is classification which involves two classes and often deals with problems in which an output is either true or false. A class is some category which is given to some subset of inputs which match the classes criteria. In the context of the bird example, a classification problem may have three classes for eagles, falcons, and pigeons. On the other hand regression deals with labels which are real valued, for example $\mathcal{Y} = \mathbb{R}$ when the output is the top speed of a given bird.

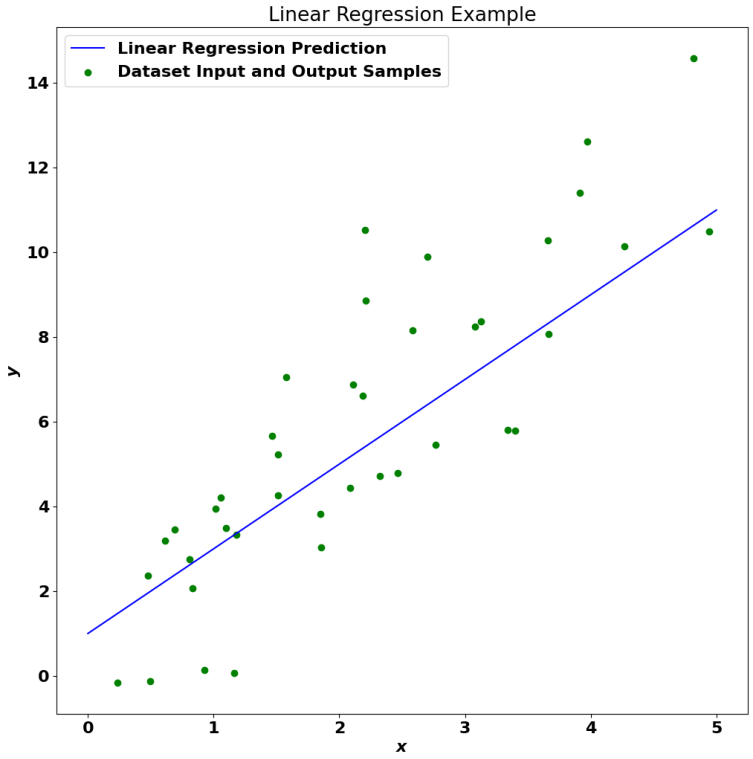


Figure 1: Example of linear regression with $\mathcal{Y} = \mathcal{X} = \mathbb{R}$ and $\mu_y = 2x + 1$.

Often a particular set of features cannot be mapped to a given class or value with complete confidence. In the bird example, birds will have different heights and wingspans that follow a probability distribution of some kind. It is possible that these distributions have some overlap, as a result some birds of a particular class may have similar heights and wingspans of another class. Further, the features provided may just not be enough to make unique mappings between $\mathbf{x}$ and $\mathbf{y}$ in the context of supervised learning. Birds of the same height and wingspan may have different top speeds. Even in the case of images it may be difficult to distinguish a bird in flight due to image resolution, weather, or any number of factors. As a result the outputs of a machine learning algorithm are inherently probabilistic. To include probability in classification the output of a machine learning model $f(\mathbf{x})$ is typically a probability distribution given the features $p(y|\mathbf{x})$. To integrate probability into regression the output is typically interpreted as the mean of the distribution conditioned on a given set of features $\mathbb{E}[y|\mathbf{x}]$.

A straight forward way to build $f(\mathbf{x})$ is with a real-valued mathematical function. One possible function is a combination of a linear function and a translation, also called an affine function.

$$f(\mathbf{x}; \mathbf{w}, b) = \mathbf{w}^\intercal \mathbf{x} + b : \mathbf{x} \in \mathbb{R}^{D \times 1}, \mathbf{w} \in \mathbb{R}^{1 \times D}, b \in \mathbb{R}$$

This is sufficient for regression assuming a linear relationship between $\mathbf{x}$ and $\mathbf{y}$. Such a model is called a linear regression model. A dataset of input, output regression pairs along with a possible linear regression solution for $\mathcal{Y} = \mathcal{X} = \mathbb{R}$ is given in Figure 1.

Similarly, a real-valued function can be used to perform binary classification by representing a surface which separates two distinct groups. In the context of machine learning the surface boundary is also called the decision boundary. Such a boundary can be formed using the affine function as in linear regression. However,

the resulting function output would have a domain of $\mathbb{R}$. This issue can be resolved by applying the sigmoid function given

$$\sigma(a) \triangleq \frac{1}{1+e^{-a}} : a \in \mathbb{R}$$

which will collapse the range of output values to $[0, 1]$. Applying the sigmoid to the affine function gives a model which can output a probability for binary classification.

$$p(y|\mathbf{x}; \mathbf{w}, b) = \begin{cases} 1 - \sigma(f(\mathbf{x}; \mathbf{w}, b)) & y = 0 \\ \sigma(f(\mathbf{x})) & y = 1 \end{cases}$$

Such a model is called a binary logistic regression model. An example of binary logistic regression is given in Figure 5. To deal with $N > 2$ classes several surfaces can be defined using the following linear function:

$$f(\mathbf{x}; \mathbf{W}, \mathbf{b}) = \mathbf{W}\mathbf{x} + \mathbf{b} : \mathbf{W} \in \mathbb{R}^{C \times D}, \mathbf{b} \in \mathbb{R}^{C \times 1}$$

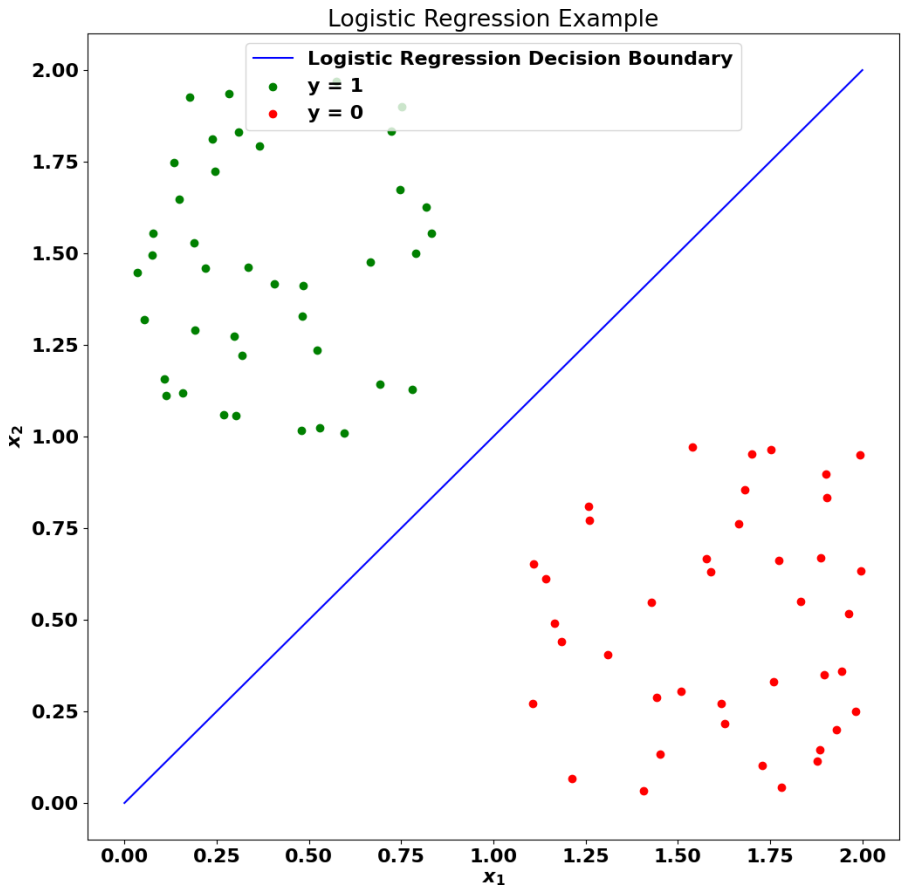


Figure 2: Example of binary logistic regression with $\mathcal{X} = \mathbb{R}^2$, $\mathcal{C} = \{0, 1\}$ with means $\mu_{\mathbf{0}} = [1.5, 0.5]^\top$ and $\mu_{\mathbf{1}} = [0.5, 1.5]^\top$.

While the sigmoid function can scale individual outputs to $[0, 1]$ it is insufficient to create a probability distribution for $N > 2$ since it does not guarantee that the sum of probabilities will equal one. An analog to sigmoid for $N > 2$ classes is the softmax function which can be used to deal with this problem.

$$\sigma(\mathbf{a})_c \triangleq \frac{e^{a_c}}{\sum_{i=1}^{C} e^{a_i}}$$

Using softmax on the above gives a model for multiclass logistic regression.

$$p(y|\mathbf{x}; \mathbf{W}, \mathbf{b}) = \sigma(f(\mathbf{x}; \mathbf{W}, \mathbf{b}))_y$$

Multiclass logistic regression is also often referred to as multinomial logistic re-

gression. A graphical representation of linear regression, binary logistic regression, and multinomial logistic regression is given in Figure 3.

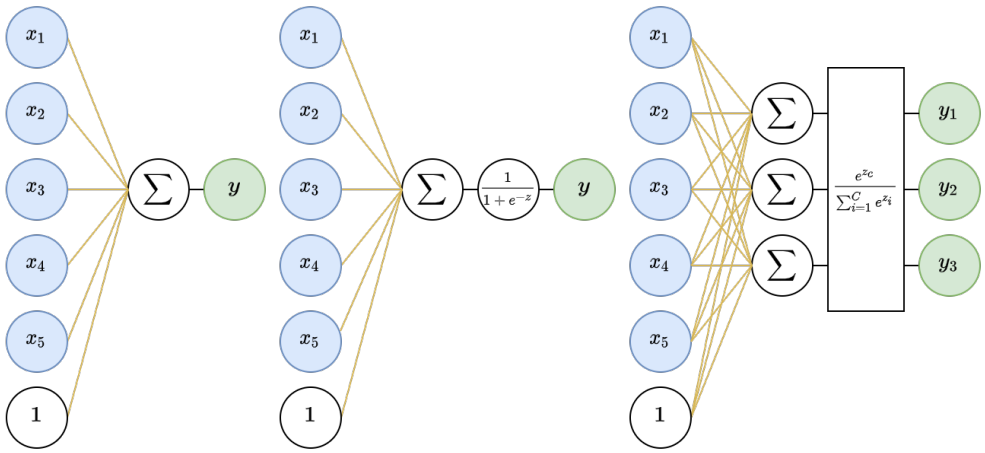


Figure 3: Graphical representation of linear regression (left), binary logistic regression (center), and multinomial logistic regression (right) with $\mathcal{X} = \mathbb{R}^5$. Blue nodes represent input values and green nodes represent outputs. Yellow edges represent multiplication by a weight value while black edges represent multiplication by 1. Note biases are represented using a weight connected to a fixed value of 1.

2.1.2 Neural Networks

Both linear and logistic regression are limited to problems where relationships between inputs and outputs are linear. To form more complex decision boundaries many linear decision boundaries can be combined. Consider binary logistic regression to form a single decision boundary. Such a model will output a probability value, taking on a value of 0.5 on the decision boundary. N of such functions $\{\sigma(f(\mathbf{x}; \mathbf{w}_1, b_1)), \sigma(f(\mathbf{x}; \mathbf{w}_2, b_2)), \ldots, \sigma(f(\mathbf{x}; \mathbf{w}_N, b_N))\}$ can be applied to form different decision boundaries which have some area of overlap. In other words, there is some region $\tilde{\mathbf{x}} \in \mathcal{X}$ for which $f(\tilde{\mathbf{x}}; \mathbf{w}_i, b_i) \geq 0$ for all $i \in \{1, 2, \ldots, N\}$. To combine the results of each model, the outputs of each $\sigma(f(\mathbf{x}; \mathbf{w}_i, b_i))$ can be formed into a new vector $\mathbf{x}_1 = [\sigma(f(\mathbf{x}; \mathbf{w}_1, b_1)), \sigma(f(\mathbf{x}; \mathbf{w}_2, b_2)), \ldots, \sigma(f(\mathbf{x}; \mathbf{w}_N, b_N))]$ and passed into another logistic regression model defined by $\sigma_1(f(\mathbf{x}_1; \mathbf{w}_1^1, b_1^1))$ to form

the final decision. A simple non-linear decision boundary is shown in Figure 4 with $N = 2$ decision boundaries used for the solution. The resulting $\mathbf{x}_1$ of that set of data is then plotted with a possible decision boundary for $f(\mathbf{x}_1; \mathbf{w}_1^1, b_1^1)$. By transforming $\mathbf{x}$ into $\mathbf{x}_1$ it is possible to solve the presented problem using logistic regression. The process of transforming features into new features which are more readily interpreted is called feature extraction.

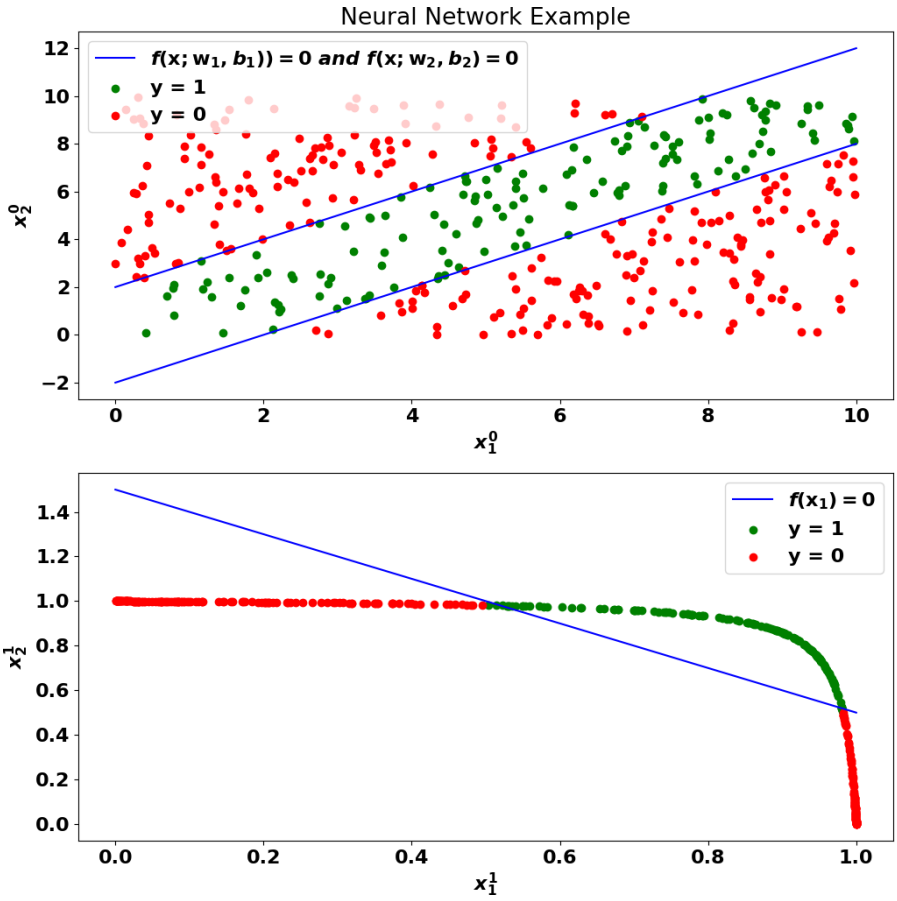


Figure 4: Example of decision boundaries for a two-layer MLP for $\mathcal{X} = \mathbb{R}^2$, $\mathcal{C} = \{0, 1\}$. Two hidden neurons are used to form two linear decision boundaries (top). The resulting transformation is then linearly separable allowing binary classification at the output neuron (bottom).

Such a combination of logistic regression models can be interpreted as a simple neural network called a multi-layer perceptron or MLP. In the context of neural networks each $\sigma_i(f(\mathbf{x}_i; \mathbf{w}_j^i, b_j^i))$ is called a neuron, with $\mathbf{w}_j^i$ being called

the weight and b_j^i being called the bias. In addition, the function $\sigma_i(\cdot)$ applied to the affine operation, sigmoid in the case of logistic regression models, is called the activation function. Activation functions other than sigmoid are commonly used in deep learning models due to problems encountered with the sigmoid function when performing learning. MLPs are often described in layers. In the above example a two-layer MLP is described where $\mathbf{x}$ is called the input layer, $\mathbf{x}_1$ is called a hidden layer, and $\mathbf{x}_2 = [\sigma_1(f(\mathbf{x}_1; \mathbf{w}_1^1, b_1^1))]$ is called the output layer. More hidden layers could be added after $\mathbf{x}_1$ to create $\mathbf{x}_2, \mathbf{x}_3, \ldots, \mathbf{x}_M$ with $\mathbf{x}_M$ being the output layer. Similar to multi-class logistic regression when dealing with multiple classes, each layer of neurons $\mathbf{x}_i = [\sigma_{i-1}(f(\mathbf{x}_{i-1}; \mathbf{w}_1^{i-1}, b_1^{i-1})), \sigma_{i-1}(f(\mathbf{x}_{i-1}; \mathbf{w}_2^{i-1}, b_2^{i-1})), \ldots, \sigma_{i-1}(f(\mathbf{x}_{i-1}; \mathbf{w}_N^{i-1}, b_N^{i-1}))]$ can be described using a single weight matrix and bias vector

$$\mathbf{x}_i = s_{i-1}(\mathbf{x}_{i-1}) = \sigma_{i-1}(f(\mathbf{x}_{i-1}; \mathbf{W}^{i-1}, \mathbf{b}^{i-1}) : \mathbf{W}^{i-1} \in \mathbb{R}^{N_i \times N_{i-1}}, \mathbf{b}^{i-1} \in \mathbb{R}^{N_i}$$

$$f(\mathbf{x}; \mathbf{W}, \mathbf{b}) = \mathbf{W}\mathbf{x} + \mathbf{b}$$

where $\sigma_{i-1}(\cdot)$ is the specific activation function used for layer $i-1$. The above is more generally called a linear layer, fully connected layer, or dense layer. An entire MLP of depth M can then be given as

$$f(\mathbf{x}; \theta) = s_{M-1}(s_{M-2}(\ldots s_1(s_0(\mathbf{x}))\ldots))$$

where θ represents all weights and biases in the network. The output layer $\mathbf{x}_M$ can be interpreted as a logistic regression model either using sigmoid or softmax activations for binary or multiclass classification respectively. Other activations, such as the identity function, can be used on the output layer for regression problems.

Historically, MLPs originate from the perceptron in the 1950s, which is essentially a neuron with a unit step as an activation function [3]. It has been shown

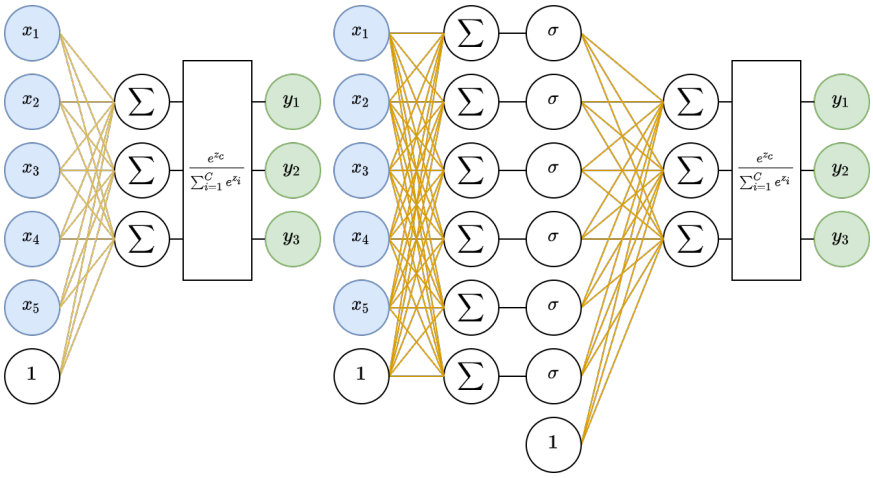


Figure 5: Graphical representation of multinomial logistic regression (left), and a 2 layer MLP classifier (right) with $\mathcal{X} = \mathbb{R}^5$ and $\mathcal{Y} = \{1, 2, 3\}$. The MLP presented has 1 hidden layer with 6 neurons. Blue nodes represent input values and green nodes represent outputs. Yellow edges represent multiplication by a weight value while black edges represent a multiplication by 1. Note biases are represented using a weight connected to a fixed value of 1.

that any solution can be found with a MLP containing a single hidden layer [4]. However, the number of hidden neurons required can grow exponentially for certain problems [5]. On the other hand MLPs and other neural networks which use more layers rather than having larger (wider) layers have shown great promise [6]. These deep neural networks can be interpreted as producing a solution hierarchically, using each layer to produce more general features. Deep neural networks have become a major interest area in recent times.

2.1.3 Stochastic Gradient Descent and Backpropagation

Most methods used for training neural networks, especially deep models, are gradient based. Let $f(\mathbf{x}, \theta)$ be a deep neural network which takes a set of features $\mathbf{x}$ and has parameters θ. Using a sample $(\mathbf{x}_i, \mathbf{y}_i)$ from a training dataset $\mathcal{D}$ a prediction $\hat{\mathbf{y}}_i = f(\mathbf{x}_i, \theta)$ can be made. Gradient based methods rely on creating a single scalar value from a model's prediction using what is called the loss function

$l(\mathbf{y}_i, \hat{\mathbf{y}}_i) = l(\mathbf{y}_i, f(\mathbf{x}_i, \theta)) = l(\theta; \mathbf{x}_i, \mathbf{y}_i) \in \mathbb{R}$. The loss function is intended to measure the models performance by producing a larger value when a prediction is more "incorrect" and a value close to 0 when it is correct. For example, in regression, a common loss function is L2 distance.

$$l_{\text{L2}}(\mathbf{y}_i, \hat{\mathbf{y}}_i) = \frac{1}{2}||\mathbf{y}_i - \hat{\mathbf{y}}_i||_{\text{L2}}$$

In binary classification binary cross entropy (BCE) loss is common, where $\hat{y}_i$ is a probability, usually coming from a sigmoid activation.

$$l_{\text{BCE}}(y_i, \hat{y}_i) = [y_i \log \hat{y}_i + (1 - y_i) \log(1 - \hat{y}_i)]$$

To generalize to $N > 2$ classes cross entropy (CE) loss is used. In general cross entropy loss is the negative log probability of the probability of the correct class.

$$l_{\text{CE}}(y_i, \hat{\mathbf{y}}_i) = -\log((\hat{\mathbf{y}}_i)_{y_i})$$

Using a single input $\mathbf{x}_i$ the loss function can be seen as a function of all model parameters θ. Stochastic Gradient Descent (SGD) *optimizes* a model by taking a step $\Delta\theta$ which reduces the value of $l(\theta; \mathbf{x}_i, \mathbf{y}_i)$. This is done in a series of steps $t = 1, 2, \ldots$ at which a different sample of data $(\mathbf{x}_{i(t)}, \mathbf{y}_{i(t)})$ is used to compute the gradient $\nabla_\theta l(\theta; \mathbf{x}_{i(t)}, \mathbf{y}_{i(t)})_t$. The step $\Delta\theta$ is then chosen to move in the direction opposite the gradient. SGD is described by the update rule

$$\theta_{t+1} = \theta_t - \alpha \nabla_\theta l(\theta; \mathbf{x}_{i(t)}, \mathbf{y}_{i(t)})_t$$

where α is the learning rate, a value which determines how far to step in the direction opposite the gradient. Each step of SGD disregards all elements in the dataset except for the one used. To optimize with respect to all data points batch

gradient descent can be used which is performed the same way, but calculates the gradient for every datapoint in a single iteration. To compute a final gradient update, all computed gradients are averaged.

$$\theta_{t+1} = \theta_t - \frac{\alpha}{N} \sum_{i=1}^{N} \nabla_\theta l(\theta; \mathbf{x}_i, \mathbf{y}_i)_t$$

Batch gradient decent more accurately matches the full dataset as it optimizes with respect to all data samples. However, batch gradient descent takes significantly more time to compute per iteration. To provide a compromise mini-batch gradient descent is commonly used which uses a small collection of data samples at each iteration.

To actually compute the gradients of each parameter in θ a method known as backpropagation is used. Backpropagation is essentially another term for chain-rule from calculus and is the process of finding the partial derivatives of the loss function with respect to all trainable model parameters. Backpropagation is described as such because all gradients are dependent on gradients from deeper in the network, meaning gradient computation moves backward with respect to the forward pass.

2.2 CNNs and Image Classification

MLPs encounter a number of issues when applied to images. For one, they do not scale well. High resolution images typically contain millions of pixels so the weights for MLPs can easily become on the order of billions for image tasks. Also, they lack what is known as inductive bias for image tasks. An inductive bias is a set of assumptions a machine learning model makes prior to learning. This is helpful to both reduce computation and enhance accuracy. For images, many transformations can be applied that are reversible and often represent the same information. For example, rotating, scaling, and translations applied to an image

will result in a similar set of features. However, since images are made of discrete pixels a translation is the only operation of the above which can trivially be made and result in the same set of features. A transitionally-invariant inductive bias is what forms the basis of convolutional neural networks (CNNs).

2.2.1 Convolutions

CNNs can perform typical machine learning tasks such as classification and regression. However, unlike an MLP they take a structured input such that $\mathcal{X} = \mathbb{R}^{C \times H \times W}$ where $\mathbf{x}$ is often an image. If $\mathbf{x}$ is an image H and W are the height and width of the image, and C is the number of color channels, typically 3 for RGB images.

Convolutional layers form the main building block of CNNs. Convolutional layers are similar to linear layers but are different in two main aspects. First, the weight is a tensor $\mathbf{w} \in \mathbb{R}^{C \times W_k \times H_k}$. This weight is often also called the kernel. Secondly, when applied the convolutional layer only considers a small portion of the input image equal to the shape of the kernel. To generate an output this weight is applied in a sliding window fashion across the whole input image to produce what is essentially a new transformed image called a feature map. The value at location x, y of the output feature map $\mathbf{y}$ given $y_{x,y}$ for input $\mathbf{x}$ can be given as follows.

$$y_{y,x} = \sigma(\sum_{i=0}^{C-1} \sum_{j=0}^{H_k-1} \sum_{k=0}^{W_k-1} \mathbf{x}_{i,y+j,x+k} \cdot \mathbf{w}_{i,j,k} + b)$$

Kernel sizes are usually small and odd numbered in size such as 3×3, 5×5 or 7×7. Also, to preserve height and width $\mathbf{x}$ is often bordered with empty values. It is also common to increase the step size performed, called the stride, to be greater than one. Incorporating stride S into the above equation gives the following.

$$y_{y,x} = \sigma(\sum_{i=0}^{C-1} \sum_{j=0}^{H_k-1} \sum_{k=0}^{W_k-1} \mathbf{x}_{i,S\cdot y+j,S\cdot x+k} \cdot \mathbf{w}_{i,j,k} + b)$$

If D convolutional layers are used, the output then has depth D giving

$$y_{z,y,x} = \sigma(\sum_{i=0}^{C-1} \sum_{j=0}^{H_k-1} \sum_{k=0}^{W_k-1} \mathbf{x}_{i,S\cdot y+j,S\cdot x+k} \cdot \mathbf{w}_{z,i,j,k} + b) : \mathbf{w} \in \mathbb{R}^{D\times C\times H_k\times W_k}$$

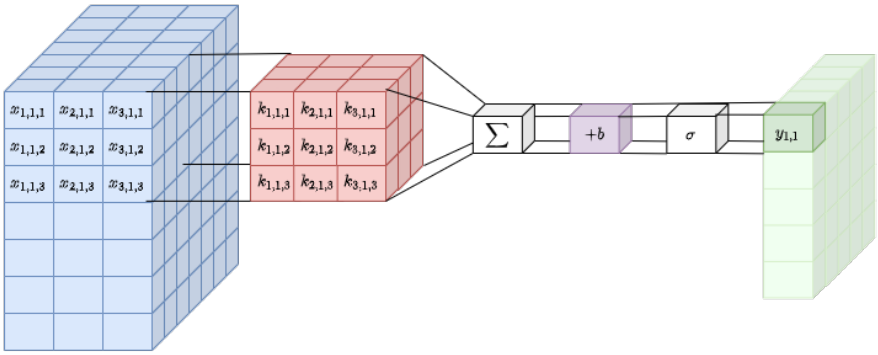


Figure 6: Visualization of a $3 \times 3 \times 3$ convolution applied to a $3 \times 7 \times 7$ input. Each output value (green) corresponds to the same operation applied to different locations in the input image (blue).

The amount of information captured for a single convolution layer depends on the kernel size. Alternatively, several convolutions can be applied consecutively. By applying several consecutive convolutional layers each layer will absorb spatial information of some area in the previous feature map. As a result with each additional convolutional layer a larger area of the original input is processed as demonstrated in Figure 7. The area of the input which a single feature map pixel considers at a given layer is called the receptive field.

2.2.2 Pooling

In addition to convolutions, CNNs typically have many other operations which are applied to feature maps. Pooling layers are another common layer which function by down-sampling feature maps. Specifically for some pooling operation

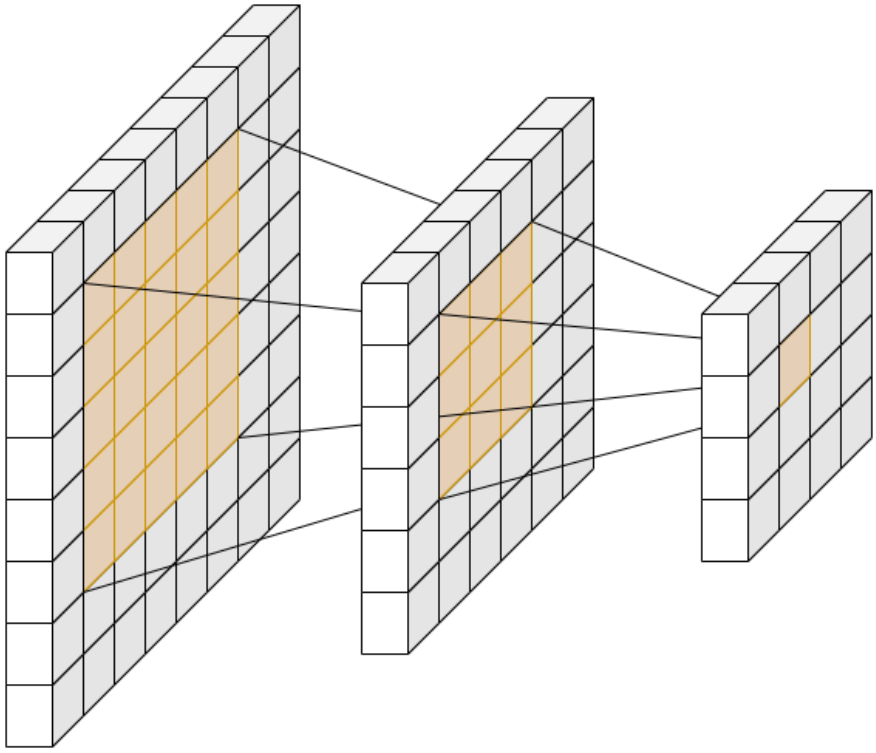


Figure 7: Visualization of two cascaded convolutions with stride 1 and a kernel size of 3×3.

$\mathbf{y} = f(\mathbf{x})$ with $\mathbf{x} \in \mathbb{R}^{C \times H_i \times W_i}, \mathbf{y} \in \mathbb{R}^{C \times H_o \times W_o}$ the input and output spatial sizes change such that $W_o < W_i, H_o < H_i$. Pooling is useful for condensing information where global information is needed such as in image classification. Pooling may also be used to reduce computation. One common pooling method is Max Pooling in which the maximum is taken over the input kernel

$$y_{z,y,x} = \max_{i=0,1,2,\ldots,H_k-1} \max_{j=0,1,2,\ldots,W_k-1} x_{z,Sy+i,Sx+j}$$

Max Pooling was seen commonly in earlier CNN models. Global Average Pooling is another common operation which takes an average of an entire feature map spatially to produce a fixed size vector.

$$y_{z,y,x} = y_z = \frac{1}{H_k W_k} \sum_{i=0}^{H_k-1} \sum_{j=0}^{W_k-1} \mathbf{x}_{z,i,j}$$

Global Average Pooling is commonly used to produce a vector which can be used for other machine learning methods such as MLPs.

2.2.3 ReLU and Other Deep CNN Methods

One last major building block for deep CNNs are ReLU and related activation functions [7]. A common issue in deep learning is known as the vanishing gradient problem. When dealing with many cascaded linear layers or convolutions the resulting gradient for any given layer is a function of the product of preceding layer activation functions. Assuming an MLP the loss with respect to layer l as a function of the gradient with respect to some deeper layer L can be given

$$\nabla_{\mathbf{x}_L} l = \nabla_{\mathbf{x}_L} L \nabla_{\mathbf{z}_{L-1}} \mathbf{x}_L \nabla_{\mathbf{x}_{L-1}} \mathbf{z}_{L-1} \nabla_{\mathbf{z}_{L-2}} \mathbf{x}_{L-1} \dots \nabla_{\mathbf{z}_l} \mathbf{x}_{l+1} \nabla_{\mathbf{x}_l} \mathbf{z}_l$$

$$\nabla_{\mathbf{x}_L} l = \nabla_{\mathbf{x}_L} L \sigma'(\mathbf{z}_{L-1}) \nabla_{\mathbf{x}_{L-1}} \mathbf{z}_{L-1} \sigma'(\mathbf{z}_{L-2}) \dots \sigma'(\mathbf{z}_l) \nabla_{\mathbf{x}_l} \mathbf{z}_l$$

where $\mathbf{x}_i = \sigma(\mathbf{z}_{i-1})$. For sigmoid, the maximum value of the gradient is $1/4$ meaning there is a decrease in magnitude for each activation. The vanishing gradient problem describes the phenomenon of decreasing gradient magnitude during backpropagation which will eventually be rounded to zero for floating point arithmetic. Since the sigmoid activation results in a decrease of gradient magnitude it may increase the risk of shrinking the gradient arbitrarily with depth and causing the vanishing gradient problem. ReLU can simply be defined as $y = max(x, 0)$ and solves the above by reducing the gradient to 0 or keeping it the same as the preceeding layer during backpropagation. As a result positive-valued activations at any given layer are not reduced by the activation function gradient.

Historically CNNs can arguably be seen to originate from LeNet-5 created by Yann LeCun in the 1980s [8]. However, it wasn't until the early 2010s with

the creation of AlexNet [7] that they saw significant research interest in the form of deep convolutional neural networks. Earlier CNNs such as AlexNet, ZF [9] and VGG-16 [10] used exclusively fully connected layers, max-pooling, ReLU, and convolutional layers. However further improvements such as Batch Normalization [11], Kaiming Initialization [12], and residual connections [13] showed to have great importance for training networks on the order of hundreds of layers. ResNet [13] and DenseNet [14] are two widely cited models which use the above mentioned methods to build very deep vision models.

2.3 Object Detection

Object detection is a task in which several objects in an image are both simultaneously located and classified. More formally an object detection model takes an image $\mathbf{x}$ and produces an output $\hat{y}$ consisting of a set of object detections $\hat{y} = \{\hat{y}_1, \hat{y}_2, ..., \hat{y}_M\}$ for M detections in total. Every object detection $\hat{y}_i$ is a set $\hat{y}_i = \{\hat{c}_i, \hat{b}_i\}$ containing a label $\hat{c}_i \in \{1, 2, ..., C+1\}$ and a bounding box $\hat{b}_i = \{\hat{b}_{i,1}, \hat{b}_{i,2}, \hat{b}_{i,3}, \hat{b}_{i,4}\}$. Typically for C classes $C+1$ labels are used to account for an additional background class which is anything not labeled by the C classes. The values of the bounding box often define the center, width, and height of the smallest box which fully encloses the associated object and is aligned with the axes of the image. CNNs have shown great success in object detection tasks. CNN object detectors are often said to perform object detection in stages. Each stage provides further refinement and strengthens object detection performance.

2.3.1 One-stage Detectors

One-stage object detectors can usually be broken up into two components called the backbone and detection head. The backbone is a CNN which produces a feature map. Object detectors commonly use existing CNN classifier architectures

by removing all linear and classification layers. It is also common to use CNNs which are already pretrained on a large scale image classification dataset such as ImageNet [15]. ImageNet contains several million images from 1000 classes.

The detection head may consist of several convolutional layers, with the last layers consisting of two convolutional networks applied in parallel which are commonly referred to as the classification branch and the regression branch. Both branches consider prior bounding boxes which are called anchors. Anchors can be considered hyperparameters and are usually defined by different sizes and aspect ratios. For k anchors and C classes, the classification head outputs kC values at each feature map location, with each value containing the probability that the associated anchor contains the associated class. The regression head contains $4k$ outputs which regress with respect to each associated anchor to refine the predicted bounding box. Typically regression values are chosen to be scale-agnostic . For an anchor $A = (A_x, A_y, A_w, A_h)$ the predicted bounding box $\hat{B} = (\hat{B}_x, \hat{B}_y, \hat{B}_w, \hat{B}_h)$ given for predicted regression values $t = (t_1, t_2, t_3, t_4)$ is given by the following as follows [16].

$$\hat{B}_x = A_w t_1 + A_x$$

$$\hat{B}_y = A_h t_2 + A_y$$

$$\hat{B}_w = A_w exp(t_3)$$

$$\hat{B}_h = A_h exp(t_4)$$

The result of the detection head is two feature maps which indicate class probabilities and regression offsets for each anchor at each feature map location. A visualization of a object detector regressor with 3 anchors is given in Figure 8 Since neighboring feature map locations encapsulate similar receptive fields many

duplicate predictions may result from using raw outputs directly. To prevent duplicates post-processing is applied to remove similar detection results. Non-maximum suppression (NMS) is a common algorithm used to solve this issue as given in Algorithm 1. NMS functions by iterating bounding boxes and keeping those which have a score higher than all other bounding boxes which meet a certain IoU threshold. The IoU between bounding boxes B_1 and B_2 can be given

$$\text{IoU} = \frac{B_1 \cap B_2}{B_1 \cup B_2}$$

As a result NMS is able to remove many of the duplicates created using CNN detectors and is vital for their performance in practical applications.

One-stage detectors are often the fastest in terms of inference speed but usually lower performing in terms of accuracy compared to those with more stages [17]. The description of one-stage and two-stage detectors was originally defined in Faster RCNN [17]. RetinaNet [18] is a one-stage model which closely matches the architecture described above.

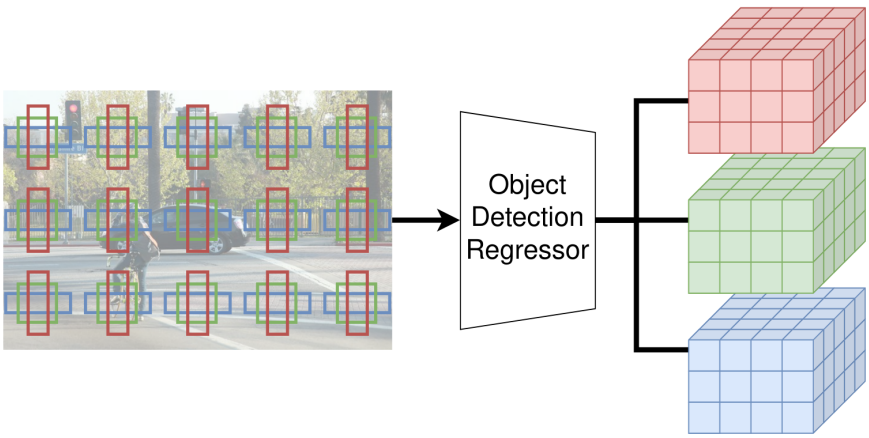


Figure 8: Visualization of a one-stage object detector bounding box regressor with three anchors. Each output feature map position is associated with a different anchor and input feature map position.

Algorithm 1 Non-Maximum Suppression (NMS) for a set of bounding boxes B and associated scores s. λ_{NMS} defines the IoU threshold for NMS.

```
procedure NMS(B,s)
    B_nms ← ∅
    for b_i ∈ B do
        keep ← True
        for b_j ∈ B do
            if IoU(b_i, b_j) > λ_NMS then
                if s_j > s_i then
                    keep ← False
                end if
            end if
        end for
        if keep then
            B_nms ← B_nms ∪ b_i
        end if
    end for
    return B_nms
end procedure
```

2.3.2 Two-stage Detectors

Two-stage detectors differ from one-stage detectors in that prior to detection the input image is analyzed to create a set of class-agnostic bounding boxes called region proposals or regions of interest (RoI). Region proposals are meant to contain areas of an image which may contain an object. Early object detectors such as RCNN [16] and Fast RCNN [19] used non-ML algorithmic methods such as SelectiveSearch [20] to obtain region proposals. Many two-stage methods are based off of Fast RCNN. Fast RCNN handles region proposals by applying an operation called RoI Pooling to create fixed size vectors. RoI pooling functions by taking the section of the feature map of size $h \times w$ associated with a region proposal and dividing it into sub windows of size $h/H \times w/W$. Max pooling is applied to each sub window to produce a fixed size output $H \times W$ which can be flattened into a fixed size vector. The resulting RoI feature vector is then passed into an MLP which has two parallel outputs with one performing classification with respect to

$C + 1$ classes and the other performing regression to create scale-invariant offsets with respect to the region proposal.

Faster RCNN [17] extends Fast RCNN by using a CNN to create region proposals called the Region Proposal Network (RPN). The RPN is the same as a one-stage detector with binary output which classifies *objectness*. Regression offsets from the RPN are then applied to their associated anchors to create a large quantity of region proposals. A final set of region proposals is then created by applying NMS and taking the top k scoring. After computing region proposals, Faster RCNN is functionally the same as Fast RCNN. A comparison of Fast RCNN and Faster RCNN architectures is given in Figure 9

2.3.3 Training Detectors

During a forward pass many anchors and regions of interest may closely match a single ground truth. During inference this issue is dealt with using NMS or some other post-processing algorithm. One way to assign bounding boxes to ground truths is by using the highest IoU per ground truth. However, this will greatly reduce the number of examples for training per image. Hundreds or thousands of anchors and RoIs may be available for training while there might only be tens of ground truth labels. Rather than producing unique assignments between predictions and ground truths multiple anchors or RoIs can be assigned to each ground truth. Some IoU threshold is chosen and all anchors and RoIs which meet this threshold are assigned to their associated ground truth. Originally $IoU_{fg} = 0.5$ for Fast RCNN and $IoU_{fg} = 0.7$ for the RPN. To provide background examples a similar process is done where IoU with all object instances must be less than some threshold, with $IoU_{bg} = 0.5$ for Fast RCNN and $IoU_{bg} = 0.3$ for the RPN. While this provides more foreground examples for training it will provide significantly more background examples still. To balance foreground and background examples

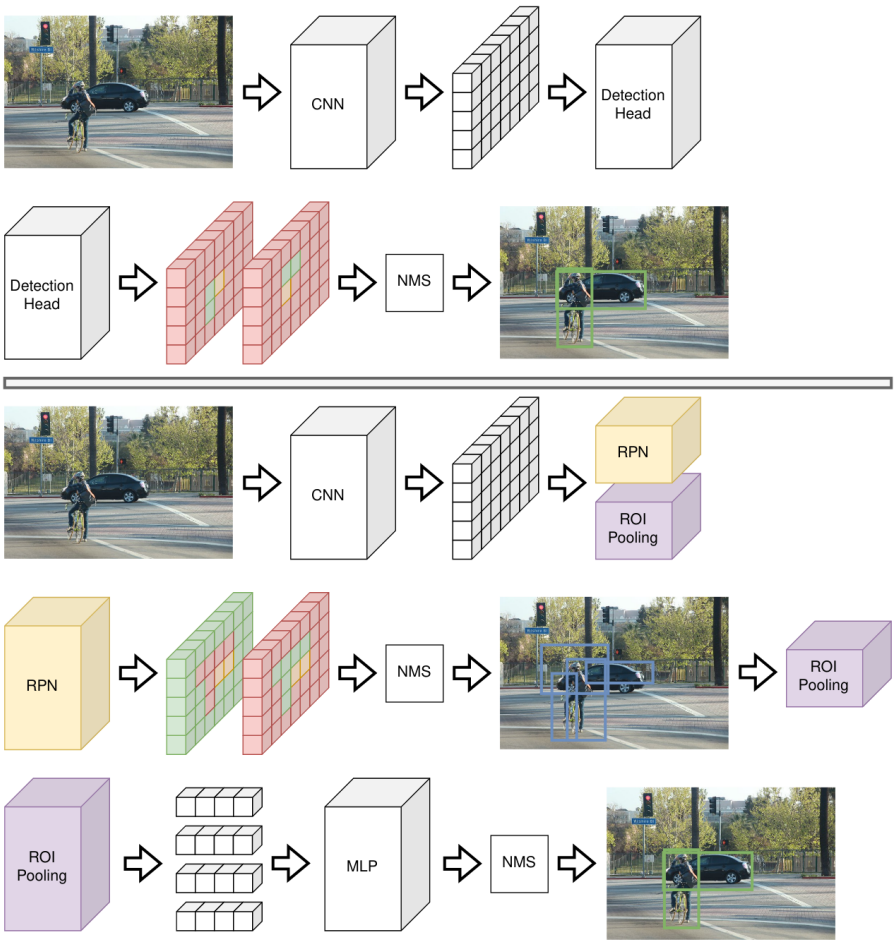


Figure 9: Comparison of one-stage (top) and two-stage (bottom) detectors. Two classes and one anchor are considered for simplicity.

a fixed number of anchors or RoIs per image is used, for example $N = 256$. Up to N_{fg} are then chosen for foreground examples, while the remaining are chosen from background examples. This image-centric fixed size and ratio sampling strategy can be seen to originally be introduced in Fast RCNN [19]. An image containing a set of example anchor assignments for $N = 32$ and $N_{fg} = \frac{N}{2}$ is given in Figure 10.

2.3.4 Feature Pyramid Networks

CNN research was initially highly focused on image classification. As a result, many CNN models are designed to gain global understanding. Deeper networks which are high performing in image classification often have lower resolution fea-

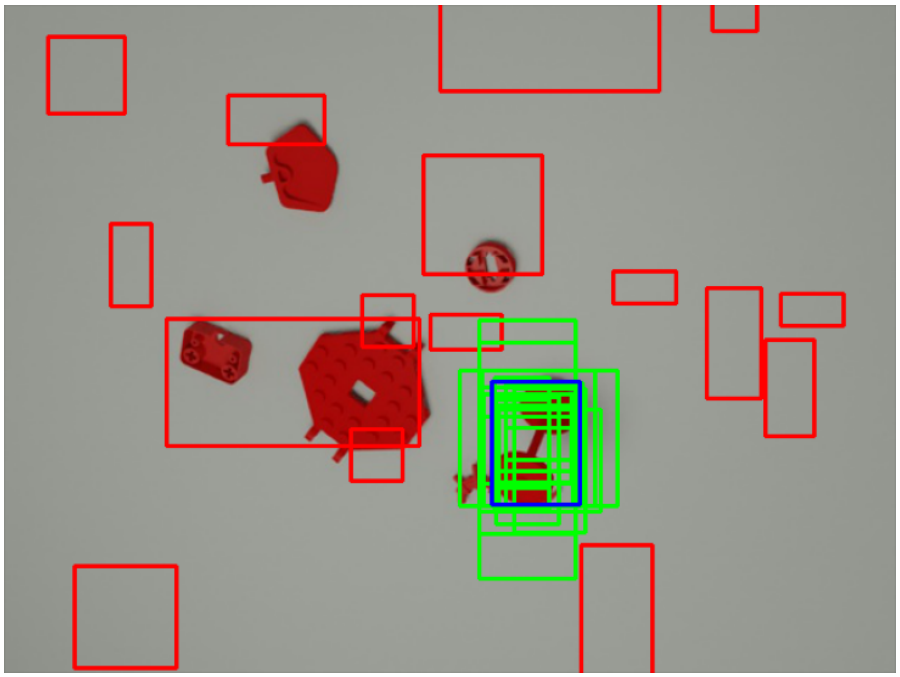


Figure 10: Bounding boxes for $N = 32$ sampled anchors. Anchors with $IoU > 0.5$ with the groundtruth (blue) are sampled for foreground examples (green) while bounding boxes with $IoU < 0.4$ are sampled for background examples (red).

.

ture maps and larger receptive fields which results in poor performance for localization. Using the intermediate layers of a backbone can help. However intermediate layer features are lower level, encapsulate a small receptive field and have weaker semantic information.

Feature Pyramid Networks (FPNs) [21] solve this issue by utilizing intermediate outputs from a CNN during the forward pass to construct higher resolution feature maps. Each intermediate layer is associated with a new feature map of the same level which contains stronger features. Specifically, starting from the backbone output, nearest neighbors upscaling is applied to create a new feature map which has the same size as the previous intermediate output. A 1×1 convolution is applied to the intermediate output and summed with the upscaled feature map as shown in Figure 11. The process is repeated for each intermediate output. The original FPN paper also considers applying convolutions to each feature pyramid

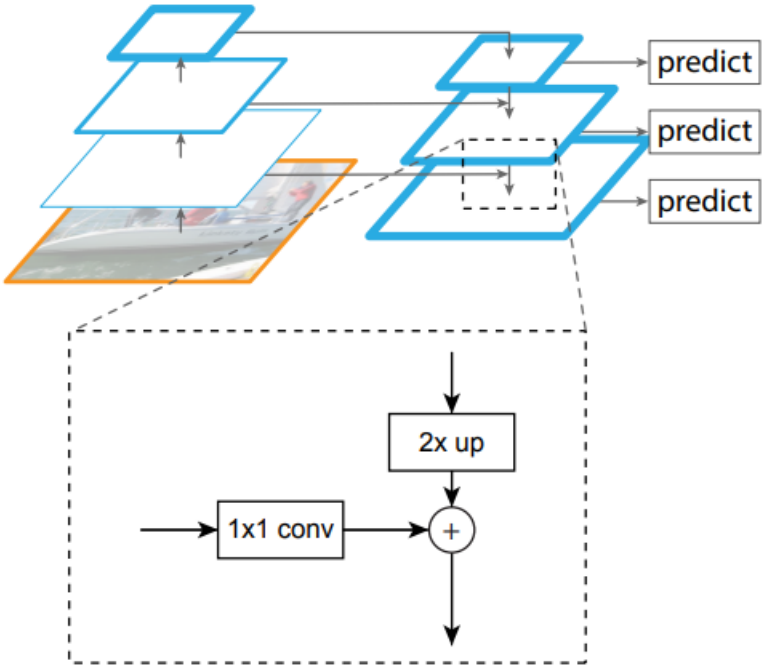


Figure 11: Feature pyramid network architecture. Aliasing convolutions are omitted. Source [21].

level to reduce aliasing caused by upsampling. The result of applying an FPN is several feature maps of different resolution which contain the same features. Each feature map level is usually associated with a different set of anchors, with higher resolution feature maps being used for small anchors and low resolution feature maps being used for large anchors. FPNs are often designed so that there is a fixed scaling ratio between pyramid levels in terms of resolution allowing the same detection head to be used for each level.

2.3.5 Image Segmentation

A common task similar to object detection is image segmentation. Image segmentation considers an image and provides a label for each pixel. Two widely investigated types of image segmentation include semantic segmentation and instance segmentation. Semantic segmentation performs pixel labeling by assign-

ing each pixel in the input image $\mathbf{x}$ to $\hat{\mathbf{s}} \in \mathbb{R}^{(C+1)\times H\times W}$ labeling C classes and background. Semantic segmentation does not distinguish different instances of objects and is useful for determining environmental attributes. On the other hand instance segmentation does consider object instances and can be considered the same as object detection but bounding boxes are replaced with some segmentation mask $\hat{\mathbf{s}}_i \in \mathbb{R}^{H\times W}$ indicating the pixels in which the associated instance is present. A comparison of different computer vision tasks including variations of image segmentation are presented in Figure 12

Mask RCNN [22] is an extension to Faster RCNN which incorporates a third parallel detection branch which predicts segmentation masks. To maintain spatial information pooled RoI features are interpreted as a feature map of fixed height and width rather than a vector. Mask RCNN also extends RoI Pooling by creating RoI Align which performs interpolation in feature map pixels to create a more accurate RoI feature map. Segmentation masks are not predicted with respect to the input image as previously described. Instead Mask RCNN predicts segmentation masks with respect to given region of interest creating a fixed scale output.

2.3.6 Measuring Object Detection Performance

Measuring the performance of object detection systems poses a few major challenges. For one, there isn't a direct association between predictions and ground truth values. It is also possible that the number of predictions made exceeds or is less than the number of ground truths. In addition, anchors and predictions are going to be overwhelmingly filled with predictions with high background probability without significant filtering. Filtering predictions also poses it's own challenge as a specific prediction score threshold may be required.

Assume that every class is considered individually. In other words, only a single class is treated as foreground and all others are treated as background.

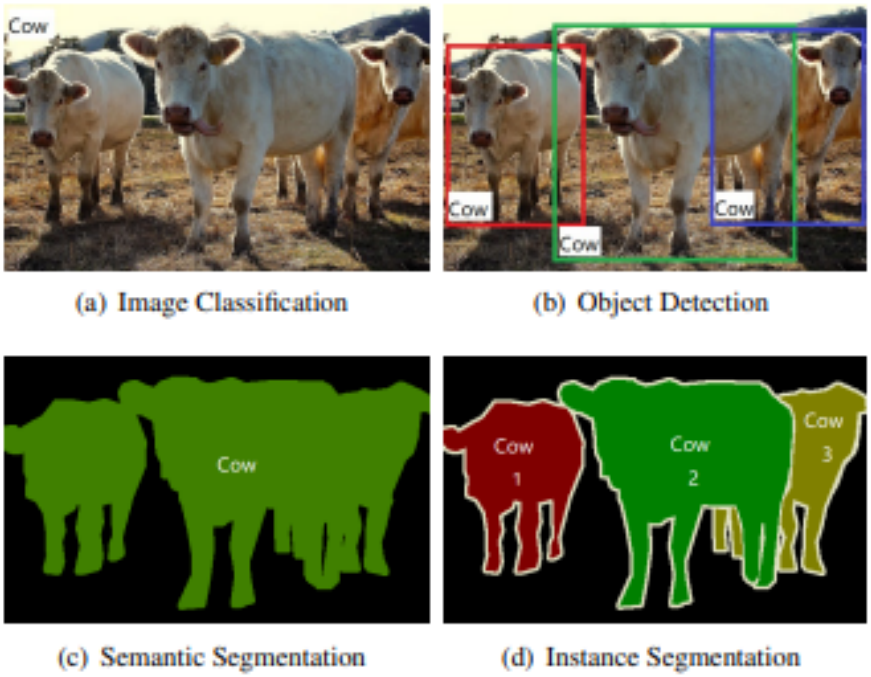


Figure 12: Comparison of common computer vision tasks. Image from [23].

Assume also that every prediction is assigned to either a ground truth bounding box of that single class or the background. Choosing a score threshold can be thought of as a problem of balancing between false negatives and false positives. Decreasing the prediction threshold will tend to result in more positive predictions and as a result less false negatives. Lowering the threshold on the other hand will tend to result in less positive results, and as a result decrease false positives.

Let $\mathcal{Y}$ be the set of all positive samples and $\hat{\mathcal{Y}}$ be the set of predictions. For a given score threshold t the set of positive predictions is given $\hat{\mathcal{Y}}_t \in \{y_i : y_i > t \text{ and } y_i \in \hat{\mathcal{Y}}\}$. To quantify the performance at a given threshold the metrics of precision and recall can be used.

$$\text{precision} = \frac{|\hat{\mathcal{Y}}_t \cap \mathcal{Y}|}{|\hat{\mathcal{Y}}|}$$

$$\text{recall} = \frac{|\hat{\mathcal{Y}}_t \cap \mathcal{Y}|}{|\mathcal{Y}|}$$

Precision measures performance in terms of false positives while recall measures performance in terms of false negatives. One way to measure performance independent of threshold is to consider all thresholds possible. For a set of N samples there are up to $N + 1$ thresholds which result in different prediction labels. The simplest way to obtain all possible sets is to sort the values in $\hat{\mathcal{Y}}$ and selecting the top $k \in [0, N]$ scoring to use as positive examples creating a total of $N+1$ sets of predictions as desired. Each prediction set can be used to compute an associated precision and recall which can be plotted to produce a precision-recall curve as shown in Figure 13. It is common alter the precision-recall curve $p(r)$ to apply interpolation as follows.

$$p(r)_{interp} = \max_{\bar{r}:\bar{r}>=r} p(\bar{r})$$

Interpolation of this kind is done to reduce the effect of small variations or 'wiggles' in the precision-recall curve [24].

To obtain a single value the precision-recall curve is integrated numerically as follows to obtain average precision or AP.

$$\text{AP} = \sum_i (r_i - r_{i-1}) p(r_i)$$

Where it is assumed recall values are sorted in non-decreasing order. AP can be computed for each class and then averaged to create a single performance metric typically called mean AP or mAP. AP solves many of the previously mentioned issues that could be encountered with other metrics, such as class imbalance. However, predictions still require assignment. To assign predictions to bounding boxes in detection IoU is used. A specific IoU threshold with a ground truth bounding boxes is required for a prediction to be considered a positive. If multiple predictions meet the specified IoU threshold with the same ground truth then typically

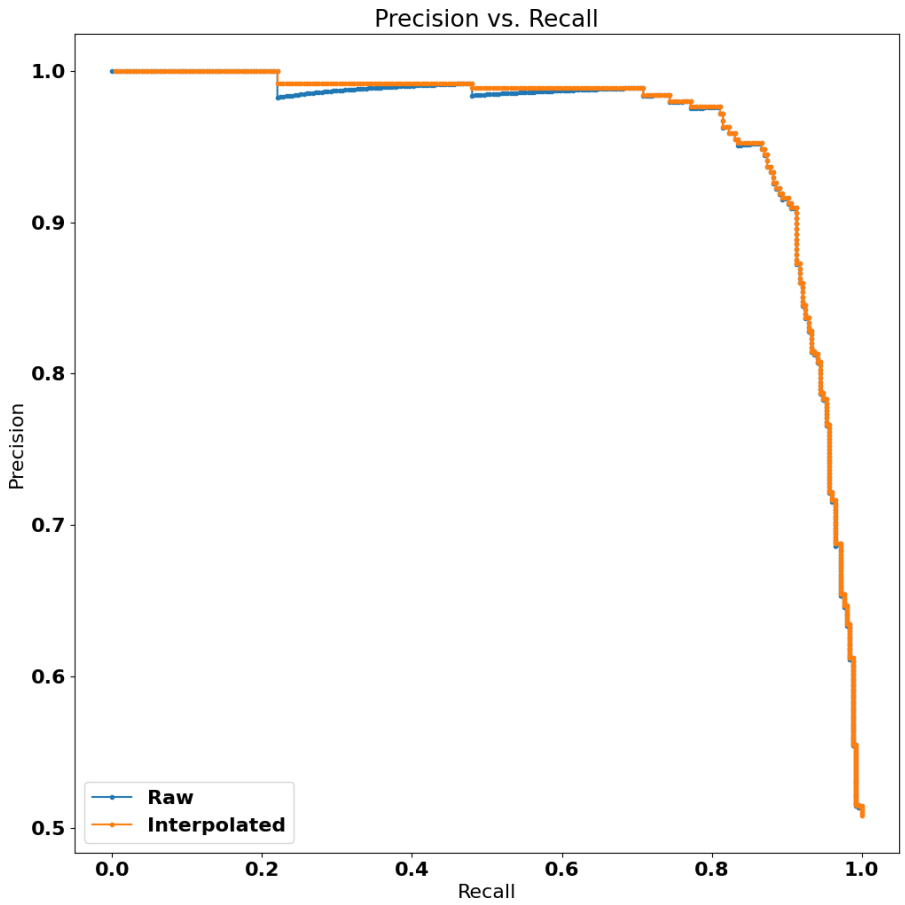


Figure 13: Precision-recall curve with interpolation created using a binary logistic regression model.

only the highest IoU prediction is marked as a positive. The remaining are considered negatives or may be assigned to other ground truths. Common IoU thresholds include 0.5 and 0.7. An example of possible assignments used for calculating AP is given in Figure 14.

2.4 Transformers

In recent times a type of deep learning model known as the transformer has been achieving high performance on various AI tasks. Transformers were originally created for natural language processing (NLP) [25] and are designed to work with sequences. Common NLP tasks performed with transformers include sequence prediction, machine translation, and text summarization. Today transformers out-

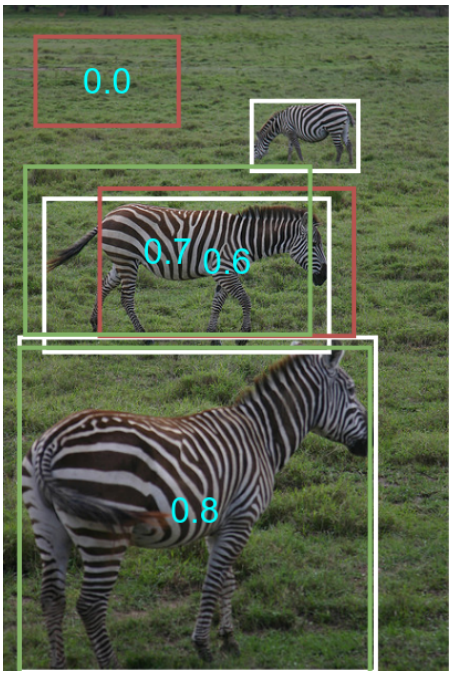


Figure 14: Example of prediction assignment for an IoU threshold of 0.5. Ground truths (white) are assigned to the prediction with the highest IoU over 0.5 (green). Predictions which do not get assigned to ground truths (red) are considered incorrect.

perform previous NLP methods such as RNNs in many benchmarks [25, 26], and more recently CNNs [27, 28] in vision benchmarks. In addition, many alterations and variations have been created, analogous to the improvements brought about by methods such as batch norm and residual connections for CNNs [29].

On a high level the original (vanilla) transformer [25] consists of two modules, an encoder and a decoder. The encoder takes the entire input sequence and creates what is called memory. The memory is then used with the decoder and some initial decoder input to perform some kind of prediction. The vanilla transform used the decoder in an auto-regressive manner, predicting the next sequence token using all previous decoder outputs and the encoder memory. This auto-regressive process is applicable to areas such as machine translation and sequence prediction.

The encoder and decoder modules themselves rely heavily on a function called

attention. Attention performs a comparison of three input sequences. A query sequence is intended extract some information about a key sequence by applying each query element to each key element. A value sequence is then used to weigh the resulting queried keys after applying normalization. As a result a global comparison of different sequence elements is performed. For sequences of length N_q and N_k with inputs $Q \in \mathbb{R}^{N_q \times d_k}$, $K \in \mathbb{R}^{N_k \times d_k}$, and $V \in \mathbb{R}^{N_k \times d_v}$ attention is given for sequence position i as

$$Attention(Q, K, V)_i = \text{Softmax}(\frac{Q_i K^T}{\sqrt{d_k}})V$$

The authors of the vanilla transformer theorized that the basic attention operation was limited in that each element could only perform a single query. To combat this an operation called multi-headed attention is proposed which projects query, key, and value elements into different subspaces. Each projection is then used to perform attention independently, the results are then concatenated and projected again to form a final result. Typically the query $Q_m \in \mathbb{R}^{N_q \times d_{model}}$, key $K_m \in \mathbb{R}^{N_k \times d_{model}}$, and value $V_m \in \mathbb{R}^{N_k \times d_{model}}$ sequences have the same depth d_{model} but may have different projection depths d_k and d_v. An image from the vanilla transformer paper is provided in Figure 15. h head multi-headed attention is given as follows.

$$MultiHeadAttn(Q_m, K_m, V_m) = Concat(head_1, ..., head_n)W^O$$

$$head_i = Attention(Q_m W_i^Q, K_m W_i^K, V_m W_i^V)$$

$$W_i^Q \in \mathbb{R}^{d_{model} \times d_k}$$

$$W_i^K \in \mathbb{R}^{d_{model} \times d_k}$$

$$W_i^V \in \mathbb{R}^{d_{model} \times d_v}$$

$$W^O \in \mathbb{R}^{hd_v \times d_{model}}$$

In the vanilla transformer multi-headed attention comes in two varieties. Multi-headed self-attention (MHSA), where query, key, and values derive from the same input, and multi-headed cross-attention (MHCA), where the query comes from a different input than the key and value.

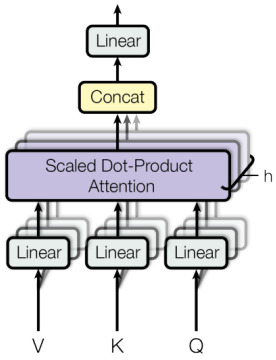


Figure 15: Multi-headed attention. Image from [25].

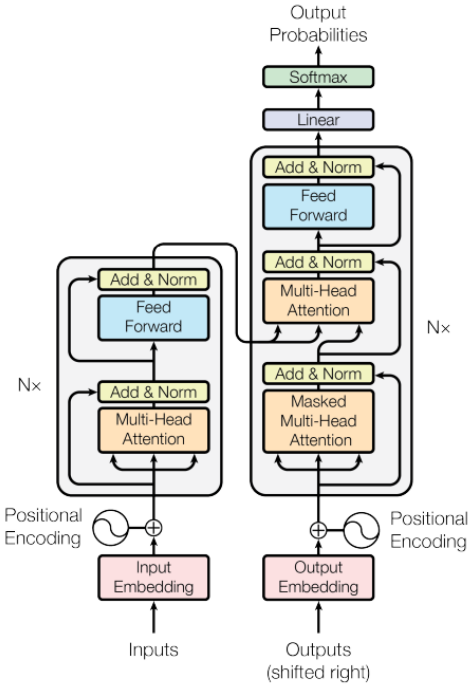


Figure 16: Architecture of the vanilla transformer. Image from [25].

The encoder consists of six architecturally identical layers each consisting of a MHSA sublayer and a simple MLP. The decoder also consists of six identical

layers with a similar architecture which differs with the presence of MHCA between MHSA and the MLP. Layers in both the encoder and decoder do not share weights. MHCA is used in the decoder by using the decoder sequence as the query and the full encoder output memory as the key and value. At the output of each sublayer in the encoder and decoder layers dropout, a residual summation, and a normalization operation are applied. One last important detail is the presence of positional encoding which is summed with the encoder and decoder input sequences prior to input into the transformer. Positional encoding allows awareness of spatial relationships, something which transformers fundamentally lack since all operations are invariant of absolute position. The full architecture of the vanilla transformer is given in Figure 16 and additional information can be found in [25].

In terms of computer vision transformers have been applied for most widely studied tasks. In most cases images are used by flattening into a sequence and relying on positional encoding to determine spatial relationships. Vision Transformer (ViT) [27] is one work which uses transformers directly for image classification. ViT showed that transformers outperform CNN methods when data is abundant. However, due to their computational complexity image pixels are not taken as direct inputs. Instead CNN feature maps are used or patches of pixels are linearly projected. DETR [28] is a foundational work for object detection and instance segmentation with transformers. Unlike CNN methods, DETR does not create duplicate outputs which need to be filtered. In addition anchors are not used, anchors are problematic in CNN detectors since performance is highly influenced by how they are defined. Both ViT and DETR have demonstrated the power transformers can have in the image domain.

List of References

[1] T. Mitchell, *Machine Learning.* McGraw-Hill Education, 1997.

[2] K. P. Murphy, *Probabilistic Machine Learning: An introduction.* MIT Press, 2022. [Online]. Available: probml.ai

[3] F. Rosenblatt, "The perceptron - a perceiving and recognizing automaton," Cornell Aeronautical Laboratory, Ithaca, New York, Tech. Rep. 85-460-1, January 1957.

[4] G. Cybenko, "Approximation by superpositions of a sigmoidal function," *Mathematics of Control, Signals, and Systems (MCSS)*, vol. 2, no. 4, pp. 303–314, Dec. 1989. [Online]. Available: http://dx.doi.org/10.1007/BF02551274

[5] R. S. Bhiksha Raj, "Lecture notes on neural nets as universal approximators," 2021.

[6] H. N. Mhaskar, Q. Liao, and T. A. Poggio, "Learning real and boolean functions: When is deep better than shallow," *CoRR*, vol. abs/1603.00988, 2016. [Online]. Available: http://arxiv.org/abs/1603.00988

[7] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds., vol. 25. Curran Associates, Inc., 2012. [Online]. Available: https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf

[8] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Backpropagation applied to handwritten zip code recognition," *Neural Computation*, vol. 1, no. 4, pp. 541–551, 1989.

[9] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," *CoRR*, vol. abs/1311.2901, 2013. [Online]. Available: http://arxiv.org/abs/1311.2901

[10] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[11] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *CoRR*, vol. abs/1502.03167, 2015. [Online]. Available: http://arxiv.org/abs/1502.03167

[12] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," *CoRR*, vol. abs/1502.01852, 2015. [Online]. Available: http://arxiv.org/abs/1502.01852

[13] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *CoRR*, vol. abs/1512.03385, 2015. [Online]. Available: http://arxiv.org/abs/1512.03385

[14] G. Huang, Z. Liu, and K. Q. Weinberger, "Densely connected convolutional networks," *CoRR*, vol. abs/1608.06993, 2016. [Online]. Available: http://arxiv.org/abs/1608.06993

[15] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 2009, pp. 248–255.

[16] R. B. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," *CoRR*, vol. abs/1311.2524, 2013. [Online]. Available: http://arxiv.org/abs/1311.2524

[17] S. Ren, K. He, R. B. Girshick, and J. Sun, "Faster R-CNN: towards real-time object detection with region proposal networks," *CoRR*, vol. abs/1506.01497, 2015. [Online]. Available: http://arxiv.org/abs/1506.01497

[18] T. Lin, P. Goyal, R. B. Girshick, K. He, and P. Dollár, "Focal loss for dense object detection," *CoRR*, vol. abs/1708.02002, 2017. [Online]. Available: http://arxiv.org/abs/1708.02002

[19] R. B. Girshick, "Fast R-CNN," *CoRR*, vol. abs/1504.08083, 2015. [Online]. Available: http://arxiv.org/abs/1504.08083

[20] J. R. Uijlings, K. E. Sande, T. Gevers, and A. W. Smeulders, "Selective search for object recognition," *Int. J. Comput. Vision*, vol. 104, no. 2, p. 154–171, sep 2013. [Online]. Available: https://doi.org/10.1007/s11263-013-0620-5

[21] T. Lin, P. Dollár, R. B. Girshick, K. He, B. Hariharan, and S. J. Belongie, "Feature pyramid networks for object detection," *CoRR*, vol. abs/1612.03144, 2016. [Online]. Available: http://arxiv.org/abs/1612.03144

[22] K. He, G. Gkioxari, P. Dollár, and R. B. Girshick, "Mask R-CNN," *CoRR*, vol. abs/1703.06870, 2017. [Online]. Available: http://arxiv.org/abs/1703.06870

[23] X. Wu, D. Sahoo, and S. C. H. Hoi, "Recent advances in deep learning for object detection," *Neurocomputing*, vol. 396, pp. 39–64, 2020.

[24] M. Everingham, L. Gool, C. K. Williams, J. Winn, and A. Zisserman, "The pascal visual object classes (voc) challenge," *Int. J. Comput. Vision*, vol. 88, no. 2, p. 303–338, jun 2010. [Online]. Available: https://doi.org/10.1007/s11263-009-0275-4

[25] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," *CoRR*, vol. abs/1706.03762, 2017.

[26] J. Devlin, M. Chang, K. Lee, and K. Toutanova, "BERT: pre-training of deep bidirectional transformers for language understanding," *CoRR*, vol. abs/1810.04805, 2018. [Online]. Available: http://arxiv.org/abs/1810.04805

[27] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, "An image is worth 16x16 words: Transformers for image recognition at scale," *CoRR*, vol. abs/2010.11929, 2020. [Online]. Available: https://arxiv.org/abs/2010.11929

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

[29] T. Lin, Y. Wang, X. Liu, and X. Qiu, "A survey of transformers," *CoRR*, vol. abs/2106.04554, 2021. [Online]. Available: https://arxiv.org/abs/2106.04554

CHAPTER 3

Few-shot Object Detection

3.1 Fine-tuning CNNs

For many applications the types of objects being considered for object detection is dynamic. For example, a robot may learn to recognize new objects or a autonomous car may need to learn to identify a new vehicle model. When training CNNs from scratch a significant amount of data is usually required. It has been shown that CNN detectors continue to increase in performance as data grows to the size of tens of millions of images and even more [1]. As a result it is impractical to consider training CNN object detectors from scratch in many of the previously stated scenarios.

Fine-tuning is a common approach used to overcome data requirements for CNN object detectors and machine learning in general when encountering new domains [2]. Many of the features in a new (novel) dataset of interest may share similarities with a large scale (base) dataset which already exists. As a result a model trained on a large scale dataset will contain features which are useful for learning the novel dataset. This makes CNNs trained on general datasets such as ImageNet [3] especially useful. The process of using knowledge from one task to enhance learning of another is known as transfer learning. Fine-tuning is the process of training an already trained (pre-trained) model on a novel dataset and is a application of transfer learning. When considering the previous task of adding a single new object class to an existing dataset fine-tuning can be used. By adding a single neuron to the output layer of the last stage classifier branch, and possibly four more to the last stage regression branch for detection, a new class can be added. Training is then done with a new dataset which contains both old classes and the new class.

3.1.1 Few-shot Object Detection

Few-shot and one-shot methods focus on learning with minimal training examples [4], with few-shot methods considering several examples and one-shot methods considering a single example. Few-shot methods are often framed by considering a model which takes two inputs, a query image $\mathbf{x} \in \mathcal{X}$ and a set of N support images $\mathbf{s} = \{\mathbf{s}_1, \mathbf{s}_2, \ldots, \mathbf{s}_N\} : \mathbf{s}_i \in \mathcal{X}$. The query image is the image on which object detection is performed on and is equivalent to the input image in a typical object detection model. On the other hand query images are labeled images which contain the object of interest to detect, also known as the target. An episode of training or testing for a few-shot model in which there are M classes with N examples each is referred to as M-Way N-Shot [5]. An example of 5-Way 1-Shot detection is given in Figure 17

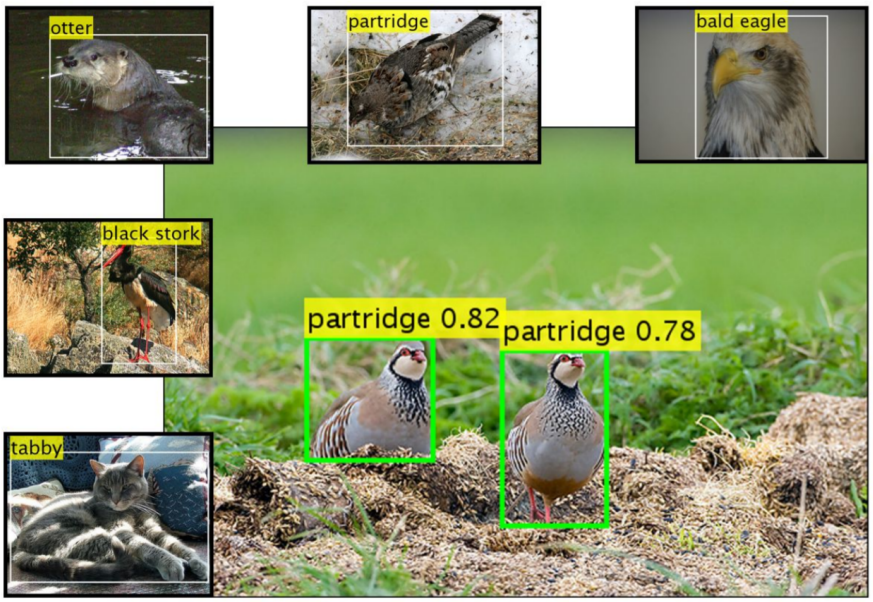


Figure 17: Example of 5-Way 1-Shot object detection. Image from [6].

3.1.2 Few-shot Object Detection with Fine-tuning

Many few-shot methods focus exclusively on the task of enhancing fine-tuning. One of the earliest attempts at few-shot object detection was LSTD [7] which focuses on fine-tuning only for target classes. LSTD utilizes an architecture which combines elements from both SSD [8] and Faster-RCNN. Bounding box regression is done based on SSD to increase robustness to object size variation. In addition, two regularization functions are proposed for few-shot fine-tuning. The first, background depression regularization, penalizes background activations when considering novel class data. The other, transfer-knowledge regularization, enhances novel class learning by incorporating knowledge of similar classes from the base dataset.

Many later works differ from LSTD in that they consider a testing scenario which includes both base classes and novel classes. Such a scenario presents a new challenge as novel feature information must be incorporated into the model without disturbing base class performance. TFA [9] finds that performance is highly improved by freezing all network weights except for the last layers of the detection head. In addition, a balanced dataset is used containing equal examples for all classes from both the base and novel datasets. Another method for maintaining performance on base classes is presented in Retentive RCNN [10]. Retentive RCNN achieves high performance by both freezing network components and adding additional RPN and detection heads. To create region proposals the objectness score of an anchor is determined by the maximum of both fine-tuned and base RPN scores. In addition, the fine-tuned detection head is trained on both novel and base classes as it was found that training on the small amount of examples present in novel class data was harmful. More recent fine-tuning works focus on various other problems. For example, MPSR [11] focuses on enhancing robustness to limited scale variance

by introducing multi-scale positive sample refinement which resizes RoI features. FSCE [12] adds a contrastive branch to the detection head to maximize inter-class distance and minimize intra-class distance.

3.2 Meta-learning Few-shot Methods

While transfer learning using fine-tuning is an effective approach for few-shot object detection in novel domains it can have issues when dealing with the task of continuously adding new classes. The first major issue is that the dataset size used for fine-tuning increases with each additional class added. If N images are required per class for training then the addition of a new class with K base classes will require $N(K+1)$ forward and backward passes for a full epoch of training. As a result, training time increases linearly with respect to the total number of classes used. If only novel classes are used for fine-tuning than a problem known as catastrophic forgetting can occur. Catastrophic forgetting describes the phenomenon by which is there is a rapid decrease in performance on base classes [13] when they are not included in training. Some work has been put into solving catastrophic forgetting [14, 15], and even specifically for object detection [16], however it still remains a fundamental problem.

It is possible to achieve few-shot fine-tuning without linear growth in training set size with respect to total class count. One possibility is the addition of a new network head for each set of novel classes added similar to Retentive RCNN [10]. However, during inference if all object classes are to be considered each detection head must be used individually increasing evaluation time linearly with respect to the number of novel class groups. In addition, each head must still be trained which requires many forward and backward passes with weight updates.

A similar strategy to that done in Retentive RCNN can be achieved with meta-learning. Meta-learning methods differ from fine-tuning methods in that

they can learn at inference time. Meta-learning few-shot object detection methods use support images as network inputs rather than using them to perform weight updates. Generally a meta-learning object detection model will extract features from both query and support images and aggregate them to perform object detection. Similar to the Retentive RCNN strategy, these methods will have linear evaluation time with respect to total classes present. However, the creation of support features differs from training in fine-tuning methods. The networks used to create support features are not necessarily as complex as the query branch. In addition, no back-propagation or weight updates are required meaning weights for meta-learning methods can be read-only and as a result be more easily integrated into hardware solutions.

3.2.1 Distance Metric Learning Methods

While many meta-learning few-shot object detectors use CNNs for feature extraction the method used for feature comparison differs significantly. Distance Metric Learning (DML) methods were some of the first which performed inference without weight updates. While DML methods are typically not considered to be meta-learning, in the context of few-shot object detection they are functionally similar. RepMet [6] is one of the first major DML methods used for few-shot object detection. RepMet functions by embedding the feature maps of target class RoIs into a vector space in which object similarity can be measured with L2 distance. In addition, RoI embeddings serve to function as modes of a multi-modal Gaussian distribution. To perform few-shot object detection the embeddings of several target class RoIs are computed. The posterior of each class is then determined by evaluating each classes associated distribution for a given query image RoI. Other similar DML methods such as NP-RepMet [17] and PNPDet [18] have been proposed.

3.2.2 Aggregating Query and Support Features

A more common approach for meta-learning few-shot object detectors involves channel-wise multiplication between query features and support features in a sliding window fashion similar to convolution. This is also often called depth-wise cross-correlation, cross-correlation, or simply correlation in the context of CNN methods. Depth-wise cross-correlation can be described for support feature F and query feature X as

$$F \in \mathbb{R}^{C \times K \times K} \quad X \in \mathbb{R}^{C \times H \times W}$$

$$Y_{c,y,x} = \sum_{j=0}^{K-1} \sum_{i=0}^{K-1} X_{c,y+j,x+i} \cdot F_{c,j,i}$$

$$Y = X \star F$$

Meta YOLO [19] and Meta RCNN [20] are early meta-learning few-shot object detection methods which use depth-wise cross-correlation. In both cases meta-learning is used to enhance fine-tuning, so these methods can be considered to be hybrids. Training is done in a similar manor to typical fine-tuning methods where training is split into pre-training on a set of base classes, and fine-tuning on a small dataset consisting of both novel and base classes. CNNs are used to create 1×1 vectors from support images which are depth-wise cross-correlated with query image features for aggregation.

Meta YOLO is based on YOLOv2 [21] which can be considered a one-stage detector while Meta RCNN is based on the two-stage Faster RCNN framework. Despite being two-stage Meta RCNN only considers feature combination after the RPN. As a result the RPN may miss regions which belong to novel classes. Some works such AttentionRPN [22] and OSWF [23] deal with this by also applying depth-wise cross-correlation to features used in the RPN input. Both of these works also do not require fine-tuning for novel detection and use more sophisticated combination methods rather than just depth-wise cross-correlation. Specif-

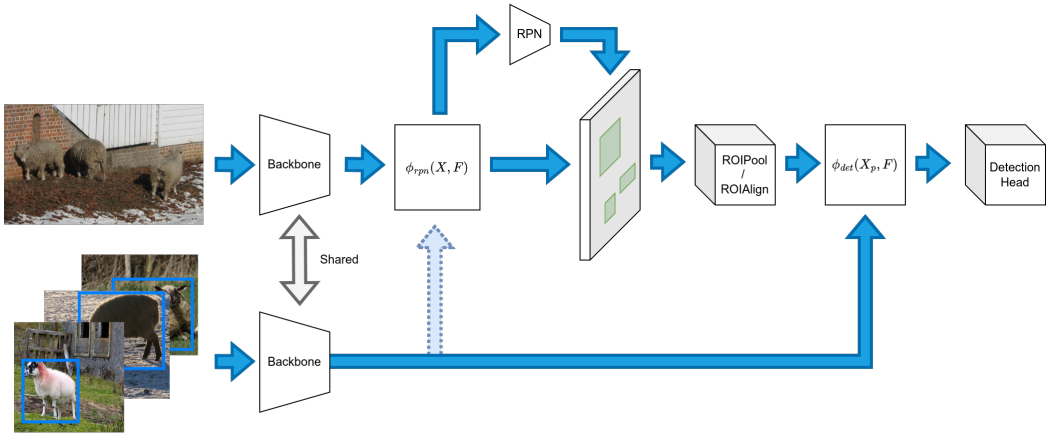


Figure 18: High level design of a two-stage meta-learning few-shot object detector. A common backbone is used to create the query image features X and support image features F. Pooled query image features are aggregated with support image features using ϕ_{det}. In addition some models aggregate query and support features prior to the RPN using ϕ_{rpn}.

ically AttentionRPN considers three different detection heads which take query and support feature maps of equal size to aggregate global information, spatially local information, and intermediate patches. OSWF has a module which functions similarly to the intermediate patch head of AttentionRPN.

Both AttentionRPN and OSWF consider the concatenation of query and support features. Other methods such as OSIS [24], FsDetView [25] and Meta Faster-RCNN [26] concatenate the result of multiple aggregation methods and incorporate a subtraction operation. Depth-wise subtraction can be described for support feature F and query feature X as follows.

$$F \in \mathbb{R}^{C \times K \times K} \quad X \in \mathbb{R}^{C \times H \times W}$$

$$Y_{c,y,x} = \sum_{j=0}^{K-1} \sum_{i=0}^{K-1} X_{c,y+j,x+i} - F_{c,j,i}$$

$$Y = X - F$$

Meta Faster RCNN differs slightly from the other two methods in that independent convolutions are applied prior to concatenation to each aggregation method result.

In addition, OSIS and FsDetView include raw query features in the concatenation while Meta Faster RCNN includes a convolution applied to concatenated raw query and support features. A comparison of aggregation methods for OSIS, FsDetView, and Meta Faster RCNN is shown in Figure 19.

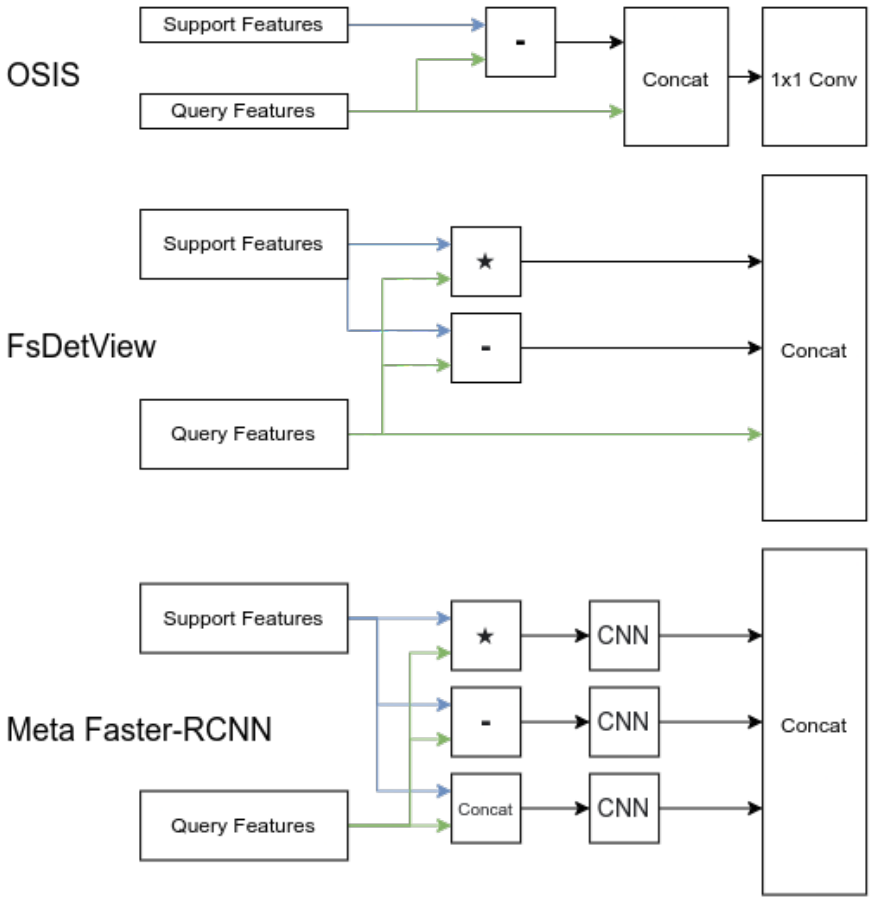


Figure 19: Meta-learning aggregation methods for OSIS[24], FsDetView[25], and Meta Faster RCNN[26].

More recently attention based methods have been increasingly used for few-shot object detection. DCNet [27] and DAnA [28] integrate aggregation methods inspired by attention. Some models also integrate transformers. AIT [29] and Meta-DETR [30] are two recently developed transformer based few-shot object detection models which have been shown to be effective.

Despite this progress few-shot methods are still limited by the fact that they require some kind of labeled data. Zero-shot methods [31] overcome this limitation by using some kind of correlated or auxiliary data related to the desired target class. Many zero shot methods use word embeddings which encode statements of natural language to create what are equivalent to supports [32, 33]. While this is useful for high level descriptions it may be less so for situations where precise object descriptions are required. An alternative representation useful for such a scenario is a 3D model.

Renders of 3D models can be created rapidly and are analogous to support images in typical few-shot object detection models. Unlike word embeddings, 3D models can precisely describe an object. In addition, 3D models are readily available for industrially manufactured components in the form of CAD files. Industrially manufactured components tend to be consistent and lack variation, as a result a 3D model makes a well suited representation. Using a zero-shot object detector which relies on 3D models for industrially manufactured components makes data collection significantly less than that required by few-shot methods which require several labeled examples of each target. In addition, renders of 3D models can be used as input into a model using meta-learning removing the need for fine-tuning. The use of synthetic target images using 3D models forms the basis for Deep Template-based Object Instance Detection (DTOID) [34].

3.3 Deep Template-based Object Instance Detection

DTOID is a deep neural network designed to take three inputs, a query image $\mathbf{x} \in \mathcal{X}$ and two types of *template* images which contain the target object named the global template $\mathbf{t}_g \in \mathcal{T}$ and the local template $\mathbf{t}_l \in \mathcal{T}$. Template images are analogous to support images in few-shot object detection but differ in that the domain of query images is not equal to that of templates. Query images are intended to be images from practical scenarios where object detection is to be employed. Template images on the other hand are derived from simple and fast rendering synthetic images created using 3D model representations of target objects. The two types of templates used in DTOID achieve different comparisons. The global template image is used to compare low level features such as textures and colors. On the other hand the local template is designed to compare high level information such as shapes and components of objects. In addition, unlike previously mentioned models, DTOID focuses on object instance detection. Object instance detection is a slightly simpler task than object detection where a query image $\mathbf{x}$ is assumed to contain a single instance of the target object. As a result the model output is always a single detection $\hat{y} = \{\hat{c}, \hat{b}\}$ containing confidence $\hat{c}$ and bounding box $\hat{b}$.

3.3.1 Architecture

DTOID consists of two modules called the correlation stage and detection stage which are analogous to the backbone and detection heads of a typical object detector. The correlation stage consists of three subnetworks: the backbone, the object attention branch (OAB) and the pose specific branch (PSB). The backbone primarly deals with query image features while the OAB and PSB create features from global and local templates respectively. The full architecture of DTOID is given in Figure 20.

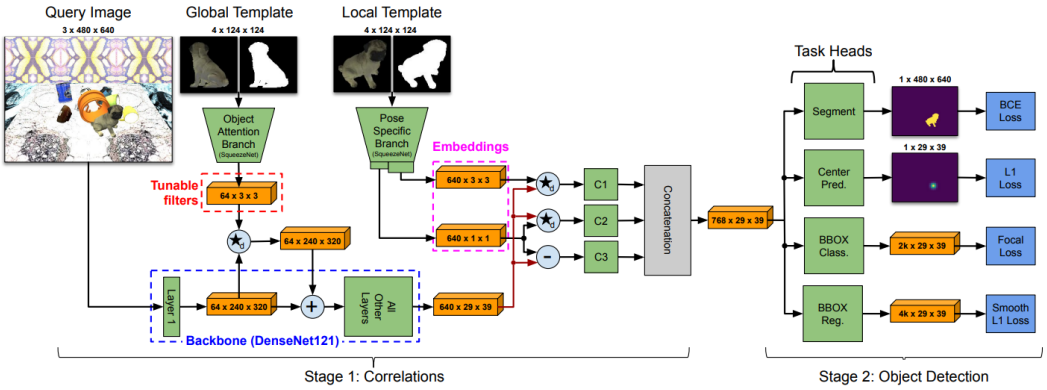


Figure 20: DTOID Architecture, image from [34].

Correlation Stage

During the forward pass a query image is processed by the first convolutional layer of the backbone. DTOID uses DenseNet121 [35] which has a first layer convolution with a kernel size of 7×7 and stride of 2. Applying global features after this convolution reduces computation considerably and allows working with low level extracted features. Global template features consist of a $C_{global} \times 3 \times 3$ tensor and are combined with the first layer query image features with depth-wise cross-correlation and a residual summation. The resulting features are then passed through the rest of the Densenet121 backbone. Local template features consist of a $C_{local} \times 3 \times 3$ and $C_{local} \times 1 \times 1$ embedding. The $C_{local} \times 3 \times 3$ embedding is intended to retain some relative spatial information about the template while the $C_{local} \times 1 \times 1$ embedding contains only global information about the local template. Both are compared with the backbone feature map using depth-wise cross-correlation. In addition, the $C_{local} \times 1 \times 1$ embedding is subtracted as this is shown to improve performance in other works [36]. The results of the three comparison operations are combined in a manor similar to Meta Faster-RCNN. Each of the three resulting feature maps have a separate convolution applied which reduces the depth from $C_{local} = 640$ to $C_{cat} = 256$. The results are then concatenated depth-wise and one

last additional convolution is applied resulting in a depth of $C_{out} = 512$.

The OAB is used to create global features and consists primarily of a modified SqueezeNet v1.1 [37] CNN. SqueezeNet is intended to be a highly lightweight CNN, minimizing the computational requirements required for creating global template features. An input template of a fixed size of 124×124 is used which is concatenated with a binary segmentation mask. The SqueezeNet is modified to allow the additional segmentation mask channel by adding an additional convolution initialized with Kaiming Initalization [38] to the input layer. All other network weights are kept from the Torchvision pretrained model. The SqueezeNet is split in half resulting in two feature maps, one of which derives from a shallower intermediate layer and the other coming from the full CNN output. These feature maps are concatenated depth-wise after downsampling the higher resolution feature map from the first half of the network. The resulting 7×7 feature map has two additional 3×3 convolutions applied giving the final $C_{global} \times 3 \times 3$ tensor used for global feature comparison.

The PSB is used to create the two local feature embeddings and has a similar architecture to the OAB. The PSB has the same architecture as the OAB up to before the last two 3×3 convolutions. To create the $C_{local} \times 3 \times 3$ embedding two additional 3×3 convolutions are applied to the 7×7 feature map given from the concatenated SqueezeNet features as done in the OAB. The $C_{local} \times 1 \times 1$ embedding is created by applying global average pooling to the 7×7 feature map given from the concatenated SqueezeNet features.

Detection Stage

The feature map created by the correlation module is used with four task heads to perform object instance detection, segmentation and center-point prediction. Segmentation and center-point prediction are considered to be auxiliary tasks and

are not considered during inference. Auxiliary tasks have been shown in previous work to enhance overall object detection performance [39]. The bounding box and regression heads are designed in a similar manner to single stage detectors such as RetinaNet by providing confidence values and regression offsets with respect to k anchors. 24 anchors are used deriving from 3 aspect ratios of $\{1:2, 1:1, 2:1\}$ and 8 scales of $\{30^2, 60^2, 90^2, 120^2, 150^2, 180^2, 210^2, 240^2\}$. The segmentation network uses a combination of vanilla convolutions and nearest neighbor upsampling to create a segmentation mask of a size equal to that of the input image. The center-point on the other hand is created using a single 3×3 convolution resulting in a low resolution heat map.

3.3.2 Datasets

Unlike few-shot detectors, DTOID considers images coming from two different domains. These domains are the query image domain $\mathcal{X}$ and template image domain $\mathcal{T}$. $\mathcal{X}$ is intended to represent a realistic and practical object detection domain. $\mathcal{T}$ on the other hand consists of images which are intended to be easy to create synthetically while containing information which is highly correlated with the associated target in the query image domain $\mathcal{X}$. Templates in $\mathcal{T}$ are created using simplified lighting techniques which can be rendered in a small fraction of a second on lower-end hardware. By comparison realistic methods such as ray tracing are much more computationally expensive.

Query Images

Unlike most of the models discussed so far the training dataset used for DTOID is created synthetically. DTOID requires object pose which is not present in many detection datasets. In addition, DTOID benefits from using a high quantity of object classes to increase generalization, something which many large scale

object detection datasets lack. Lastly, 3D models of objects are needed to generate templates. Due to these constraints the use of synthetically generated data is much more practical than creating a new dataset from real images.

For training, DTOID considers 125 objects from the 3D pose-estimation benchmarks in BOP [40]. 10,000 Query images were created to be similar to the scene of the LINEMOD [41] dataset which was being used for testing. Specifically 250 scenes were created where the objects are placed on top of a table surrounded by four walls, a floor, and a ceiling. It is stated that randomization is applied to 'the texture of the environment (walls, floor and table), lighting (placement, type, intensity and color), object materials (diffuse and specular reflection coefficients) and anti-aliasing (type and various parameters)'. This type of non-domain specific randomization is called domain randomization and is used to increase generalizing capabilities. Four to thirteen objects are sampled with 50% of simulations dropping objects with a physics simulation and the other 50% having them placed on the table so that they are standing upright. 40 images are created for each scene using 20 randomly placed camera positions. Each camera performs two renders, one using realistic physics based rendering (PBR) and another which lacks lighting and shadows. An additional 10,000 images are generated using a fast OpenGL renderer by randomly placing objects in an open space in front of a background sampled from the Sun3D dataset [42]. In total 20,000 images are generated and used for training. Example images provided from [34] are given in Figure 21.

Template Images

Template images are rendered using a fast OpenGL renderer. Objects are rendered with diffuse reflectance and ambient occlusion with lighting provided by a global ambient light and a single overhead directional light. Objects are rendered such that the largest length in the image plane is in the range of 100 to 115 pixels.

Figure 21: Images from the DTOID query image simulator, image from [34].

A border is then added to achieve the template image size of 124×124 pixels. Examples images are given in Figure 22.

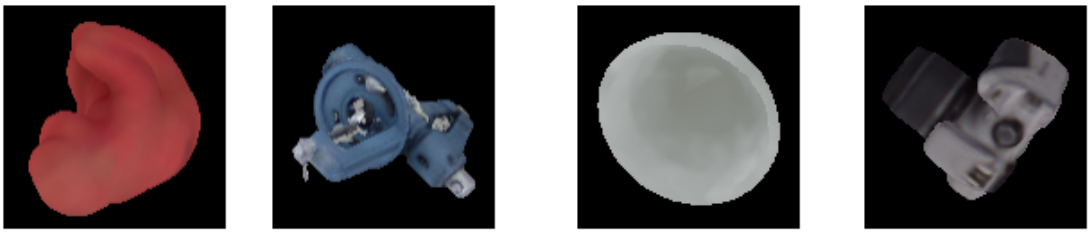


Figure 22: Example templates from DTOID, images from [34].

3.3.3 Training and Inference

During training heavy data augmentation is applied. Data augmentation is the process of applying label-preserving transformations to data samples. Data augmentation is typically applied online during training to artificially increase the amount of data available. The hue, saturation, and brightness are randomly altered in both the query and template using the object's segmentation mask. Image wide augmentation is applied such as brightness shifts, Gaussian blur, Gaussian noise, horizontal and vertical flips, and random translations and scaling. A random hue is applied to the whole image and template 50% of the time. Motion blur is then applied 20% of the time using a line kernel.

Loss $\mathcal{L}$ is computed as a weighted sum of losses from the four task heads.

$$\mathcal{L} = \lambda_{cls}\mathcal{L}_{cls} + \lambda_{reg}\mathcal{L}_{reg} + \lambda_{mask}\mathcal{L}_{mask} + \lambda_{center}\mathcal{L}_{center}$$

Classification loss $\mathcal{L}_{cls}$ is focal loss [43] given by

$$\mathcal{L}_{cls} = -\alpha(1 - p_y)^{\gamma} \log p_y$$

where α and γ are hyperparameters and p_y is the probability of the groundtruth class. Focal loss is designed to provide more weight towards predictions which have low confidence (are more incorrect). Focal loss has shown to be effective when all anchors are used during training, rather than sampling. Regression loss $\mathcal{L}_{reg}$ is smooth L1 loss [44] given by

$$\mathcal{L}_{reg} = \begin{cases} \frac{1}{2\beta}(\hat{y} - y)^2, & |\hat{y} - y| < \beta \\ |\hat{y} - y| - \frac{\beta}{2}, & \text{otherwise} \end{cases}$$

which is a combination of L1 and L2 loss resistant to outliers. β is a hyperparameter which determines the size of the region in which L2 loss is used. Segmentation mask loss $\mathcal{L}_{mask}$ and center-point loss $\mathcal{L}_{center}$ are binary cross entropy loss and L1 loss respectively.

A forward pass during training consists of a single query image, a single global template and a single local template. Global templates are chosen using a random pose while local templates are chosen to match the pose present in query images. To increase robustness to pose variations the local template pose is perturbed by selecting a random rotation vector and rotating by a random quantity. It was found in [34] that a maximum perturbation angle of 30 degrees is optimal. During inference a single global template is used to extract backbone features. 160 local templates are then used deriving from 16 different viewpoints. Each local template is applied independently in the correlation module to create a different set of detection results. The final detection result is acquired by taking the top scoring result.

Methods	Linemod (2D BBox)	Occluded Linemod (mAP)
Tjaden et al [45]	78.50	N/A
LINE-2D [46]	86.50	21.0
TDID corrs. [36]	54.37	34.13
SiamMask corrs. [47]	68.23	41.47
DTOID [34]	77.92	50.71

Table 1: Comparison of DTOID with other methods on Linemod and Occluded Linemod datasets. The 2D BBox metric is discussed in [34] while mAP is given at an IoU of 0.5.

3.3.4 Detection Performance

Data from [34] is given in Table 1 showing performance in mAP for an IoU of 0.5 on Occluded Linemod and a metric called 2D BBox on Linemod. The models shown were not aware of test-time models during training. Unlike the methods previously discussed methods compared are not few-shot focused. Instead compared methods include a mixture of classical and deep learning based models for tasks such as pose estimation and object tracking which are capable of object instance detection. Linemod and Occluded Linemod are from the BOP benchmark [40]. These sets are similar and contain the same objects, they differ mainly in that Occluded Linemod contains objects which may be only partially visible while Linemod does not. Comparing the results it can be seen that DTOID performs about 10% worse than the best model on Linemod but about 20% better than the second best on Occluded Linemod. As a result the data suggests that DTOID is best suited for environments which are less structured and in which occlusion may occur compared to other methods.

Comparing to standard object detectors DTOID is arguably weak in terms of detection performance. Performance of different Faster RCNN variants published in 2017 [39] show mAP at an IoU of 0.5 in the range of 55 to 60 on the COCO dataset. COCO is arguably more sophisticated then the scenario DTOID was tested in. For example, unlike Occluded Linemod COCO contains images in which

an object class may appear multiple times in close proximity, images with highly variable environments, significantly more intra-class variance, and about seven times more classes.

List of References

[1] C. Sun, A. Shrivastava, S. Singh, and A. Gupta, "Revisiting unreasonable effectiveness of data in deep learning era," *CoRR*, vol. abs/1707.02968, 2017.

[2] K. P. Murphy, *Probabilistic Machine Learning: An introduction*. MIT Press, 2022. [Online]. Available: probml.ai

[3] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 2009, pp. 248–255.

[4] M. Köhler, M. Eisenbach, and H. Gross, "Few-shot object detection: A survey," *CoRR*, vol. abs/2112.11699, 2021.

[5] O. Vinyals, C. Blundell, T. P. Lillicrap, K. Kavukcuoglu, and D. Wierstra, "Matching networks for one shot learning," *CoRR*, vol. abs/1606.04080, 2016.

[6] E. Schwartz, L. Karlinsky, J. Shtok, S. Harary, M. Marder, S. Pankanti, R. S. Feris, A. Kumar, R. Giryes, and A. M. Bronstein, "Repmet: Representative-based metric learning for classification and one-shot object detection," *CoRR*, vol. abs/1806.04728, 2018. [Online]. Available: http://arxiv.org/abs/1806.04728

[7] H. Chen, Y. Wang, G. Wang, and Y. Qiao, "LSTD: A low-shot transfer detector for object detection," *CoRR*, vol. abs/1803.01529, 2018.

[8] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. E. Reed, C. Fu, and A. C. Berg, "SSD: single shot multibox detector," *CoRR*, vol. abs/1512.02325, 2015.

[9] X. Wang, T. E. Huang, T. Darrell, J. E. Gonzalez, and F. Yu, "Frustratingly simple few-shot object detection," *CoRR*, vol. abs/2003.06957, 2020. [Online]. Available: https://arxiv.org/abs/2003.06957

[10] Z. Fan, Y. Ma, Z. Li, and J. Sun, "Generalized few-shot object detection without forgetting," *CoRR*, vol. abs/2105.09491, 2021.

[11] J. Wu, S. Liu, D. Huang, and Y. Wang, "Multi-scale positive sample refinement for few-shot object detection," *CoRR*, vol. abs/2007.09384, 2020. [Online]. Available: https://arxiv.org/abs/2007.09384

[12] B. Sun, B. Li, S. Cai, Y. Yuan, and C. Zhang, "FSCE: few-shot object detection via contrastive proposal encoding," *CoRR*, vol. abs/2103.05950, 2021. [Online]. Available: https://arxiv.org/abs/2103.05950

[13] M. McCloskey and N. J. Cohen, "Catastrophic interference in connectionist networks: The sequential learning problem," ser. Psychology of Learning and Motivation, G. H. Bower, Ed. Academic Press, 1989, vol. 24, pp. 109–165. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0079742108605368

[14] J. Kirkpatrick, R. Pascanu, N. C. Rabinowitz, J. Veness, G. Desjardins, A. A. Rusu, K. Milan, J. Quan, T. Ramalho, A. Grabska-Barwinska, D. Hassabis, C. Clopath, D. Kumaran, and R. Hadsell, "Overcoming catastrophic forgetting in neural networks," *CoRR*, vol. abs/1612.00796, 2016. [Online]. Available: http://arxiv.org/abs/1612.00796

[15] A. S. Zacarias and L. A. Alexandre, "Overcoming catastrophic forgetting in convolutional neural networks by selective network augmentation," *CoRR*, vol. abs/1802.08250, 2018. [Online]. Available: http://arxiv.org/abs/1802.08250

[16] K. Shmelkov, C. Schmid, and K. Alahari, "Incremental learning of object detectors without catastrophic forgetting," *CoRR*, vol. abs/1708.06977, 2017. [Online]. Available: http://arxiv.org/abs/1708.06977

[17] Y. Yang, F. Wei, M. Shi, and G. Li, "Restoring negative information in few-shot object detection," *CoRR*, vol. abs/2010.11714, 2020. [Online]. Available: https://arxiv.org/abs/2010.11714

[18] G. Zhang, K. Cui, R. Wu, S. Lu, and Y. Tian, "Pnpdet: Efficient few-shot detection without forgetting via plug-and-play sub-networks," in *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, January 2021, pp. 3823–3832.

[19] B. Kang, Z. Liu, X. Wang, F. Yu, J. Feng, and T. Darrell, "Few-shot object detection via feature reweighting," *CoRR*, vol. abs/1812.01866, 2018. [Online]. Available: http://arxiv.org/abs/1812.01866

[20] X. Yan, Z. Chen, A. Xu, X. Wang, X. Liang, and L. Lin, "Meta R-CNN : Towards general solver for instance-level low-shot learning," *CoRR*, vol. abs/1909.13032, 2019. [Online]. Available: http://arxiv.org/abs/1909.13032

[21] J. Redmon and A. Farhadi, "YOLO9000: better, faster, stronger," *CoRR*, vol. abs/1612.08242, 2016.

[22] Q. Fan, W. Zhuo, and Y. Tai, "Few-shot object detection with attention-rpn and multi-relation detector," *CoRR*, vol. abs/1908.01998, 2019.

[23] X. Li, L. Zhang, Y. P. Chen, Y. Tai, and C. Tang, "One-shot object detection without fine-tuning," *CoRR*, vol. abs/2005.03819, 2020.

[24] C. Michaelis, I. Ustyuzhaninov, M. Bethge, and A. S. Ecker, "One-shot instance segmentation," *CoRR*, vol. abs/1811.11507, 2018.

[25] Y. Xiao and R. Marlet, "Few-shot object detection and viewpoint estimation for objects in the wild," *CoRR*, vol. abs/2007.12107, 2020.

[26] G. Han, S. Huang, J. Ma, Y. He, and S. Chang, "Meta faster R-CNN: towards accurate few-shot object detection with attentive feature alignment," *CoRR*, vol. abs/2104.07719, 2021.

[27] H. Hu, S. Bai, A. Li, J. Cui, and L. Wang, "Dense relation distillation with context-aware aggregation for few-shot object detection," *CoRR*, vol. abs/2103.17115, 2021.

[28] T. Chen, Y. Liu, H. Su, Y. Chang, Y. Lin, J. Yeh, and W. H. Hsu, "Should I look at the head or the tail? dual-awareness attention for few-shot object detection," *CoRR*, vol. abs/2102.12152, 2021.

[29] D.-J. Chen, H.-Y. Hsieh, and T.-L. Liu, "Adaptive image transformer for one-shot object detection," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2021, pp. 12 247–12 256.

[30] G. Zhang, Z. Luo, K. Cui, and S. Lu, "Meta-detr: Few-shot object detection via unified image-level meta-learning," *CoRR*, vol. abs/2103.11731, 2021. [Online]. Available: https://arxiv.org/abs/2103.11731

[31] Y. Xian, C. H. Lampert, B. Schiele, and Z. Akata, "Zero-shot learning - A comprehensive evaluation of the good, the bad and the ugly," *CoRR*, vol. abs/1707.00600, 2017. [Online]. Available: http://arxiv.org/abs/1707.00600

[32] S. Rahman, S. H. Khan, and F. Porikli, "Zero-shot object detection: Learning to simultaneously recognize and localize novel concepts," *CoRR*, vol. abs/1803.06049, 2018. [Online]. Available: http://arxiv.org/abs/1803.06049

[33] A. Bansal, K. Sikka, G. Sharma, R. Chellappa, and A. Divakaran, "Zero-shot object detection," *CoRR*, vol. abs/1804.04340, 2018. [Online]. Available: http://arxiv.org/abs/1804.04340

[34] J.-P. Mercier, M. Garon, P. Giguère, and J.-F. Lalonde, "Deep template-based object instance detection," 2019. [Online]. Available: https://arxiv.org/abs/1911.11822

[35] G. Huang, Z. Liu, and K. Q. Weinberger, "Densely connected convolutional networks," *CoRR*, vol. abs/1608.06993, 2016. [Online]. Available: http://arxiv.org/abs/1608.06993

[36] P. Ammirato, C. Fu, M. Shvets, J. Kosecka, and A. C. Berg, "Target driven instance detection," *CoRR*, vol. abs/1803.04610, 2018. [Online]. Available: http://arxiv.org/abs/1803.04610

[37] F. N. Iandola, M. W. Moskewicz, K. Ashraf, S. Han, W. J. Dally, and K. Keutzer, "Squeezenet: Alexnet-level accuracy with 50x fewer parameters and <1mb model size," *CoRR*, vol. abs/1602.07360, 2016. [Online]. Available: http://arxiv.org/abs/1602.07360

[38] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," *CoRR*, vol. abs/1502.01852, 2015. [Online]. Available: http://arxiv.org/abs/1502.01852

[39] K. He, G. Gkioxari, P. Dollár, and R. B. Girshick, "Mask R-CNN," *CoRR*, vol. abs/1703.06870, 2017. [Online]. Available: http://arxiv.org/abs/1703.06870

[40] T. Hodan, F. Michel, E. Brachmann, W. Kehl, A. G. Buch, D. Kraft, B. Drost, J. Vidal, S. Ihrke, X. Zabulis, C. Sahin, F. Manhardt, F. Tombari, T. Kim, J. Matas, and C. Rother, "BOP: benchmark for 6d object pose estimation," *CoRR*, vol. abs/1808.08319, 2018.

[41] S. Hinterstoisser, S. Holzer, C. Cagniart, S. Ilic, K. Konolige, N. Navab, and V. Lepetit, "Multimodal templates for real-time detection of texture-less objects in heavily cluttered scenes," in *2011 International Conference on Computer Vision*, 2011, pp. 858–865.

[42] J. Xiao, A. Owens, and A. Torralba, "Sun3d: A database of big spaces reconstructed using sfm and object labels," in *2013 IEEE International Conference on Computer Vision*, 2013, pp. 1625–1632.

[43] T. Lin, P. Goyal, R. B. Girshick, K. He, and P. Dollár, "Focal loss for dense object detection," *CoRR*, vol. abs/1708.02002, 2017. [Online]. Available: http://arxiv.org/abs/1708.02002

[44] R. B. Girshick, "Fast R-CNN," *CoRR*, vol. abs/1504.08083, 2015. [Online]. Available: http://arxiv.org/abs/1504.08083

[45] H. Tjaden, U. Schwanecke, and E. Schömer, "Real-time monocular pose estimation of 3d objects using temporally consistent local color histograms," in *2017 IEEE International Conference on Computer Vision (ICCV)*, 2017, pp. 124–132.

[46] S. Hinterstoisser, S. Holzer, C. Cagniart, S. Ilic, K. Konolige, N. Navab, and V. Lepetit, "Multimodal templates for real-time detection of texture-less objects in heavily cluttered scenes," in *2011 International Conference on Computer Vision*, 2011, pp. 858–865.

[47] Q. Wang, L. Zhang, L. Bertinetto, W. Hu, and P. H. Torr, "Fast online object tracking and segmentation: A unifying approach," *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun 2019. [Online]. Available: http://dx.doi.org/10.1109/CVPR.2019.00142

CHAPTER 4

Methods

4.1 Changes Investigated

The potential usefulness but lack of performance in DTOID provide a reason for improvement. Three different alterations to DTOID are investigated in an attempt to improve recognition performance. The first is the alteration of a feature pyramid network (FPN) to the backbone to improve detection performance on small objects. Experiments are performed for different FPN variants. The second alteration investigates a modification of the loss function. A weight is applied to the loss which is a function of the difference between query and template object poses. As a result higher loss is applied to local templates which are more similar to the pose presented in query images. Lastly, the third alteration investigates the usage of a transformer as a replacement for depth-wise cross-correlation in the correlation module.

4.1.1 Feature Pyramid Networks

To increase performance on small objects an FPN is integrated into DTOID using the torchvision [1] utilities. The DenseNet121 backbone of DTOID can be broken up into several sections called denseblocks which vary in produced feature map resolution. Three FPN levels $\{P_4, P_3, P_2\}$ are used deriving from the outputs of denseblock 4, 3, and 2 with base anchor scales of $\{128^2, 64^2, 32^2\}$. Each FPN level has 15 anchors with 3 aspects ratios $\{1:2, 1:1, 2:1\}$ and 5 scales deriving from $\{1, 2^{1/5}, 2^{2/5}, 2^{3/5}, 2^{4/5}\}$. All FPN parameters are initialized using the pytorch default of Kaiming initialization [2] using a uniform distribution,. A large number of anchors is used to have comparable results with the basemodel in terms of anchor density while also having logarithmically progressing scales which allows the same

detection head to be used on all FPN levels. As shown in Figure 23, a vast majority of instances have a scale $\sqrt{HW}$ which lies within the range of the two highest resolution feature maps meaning anchor quantity may not have a significant bias (24 for the baseline vs 30 in the first two feature maps). An additional experiment is performed using the baseline model with the above stated 45 anchors to control for anchors selected.

The depths of convolutions used in the correlation module are changed to reduce computation and provide a more fair comparison with the baseline. Specifically each FPN level takes on an output depth of 256 rather than 640. Convolutions applied to the three depth-wise correlated feature maps deriving from the query image backbone and the local template features have a depth which is reduced from 256 to 96. The depth of the final convolution applied after concatenation is then reduced from 512 to 192. The original architecture is also tested to control for these changes.

To maintain having a single output for segmentation and center-point prediction the resulting correlation module outputs for all FPN levels are fused. Fusion is performed by upsampling all feature maps to the size of the largest. The upscaled feature maps are then aggregated to produce a single feature map. Aggregation methods investigated include the averaging of up-scaled featured maps and a concatenation followed by depth reducing convolution. In addition, different up-scaling methods are also investigated including nearest neighbor upsampling and bicubic interpolation. The methods of correlation module aggregation tested when using the FPN are displayed in Figure 24.

The anchor sampling strategy used for the FPN variants differ from the baseline. The baseline model uses focal loss [3] which is designed to use all anchors during training. Rather than using all anchors, the FPN variant uses the Fast

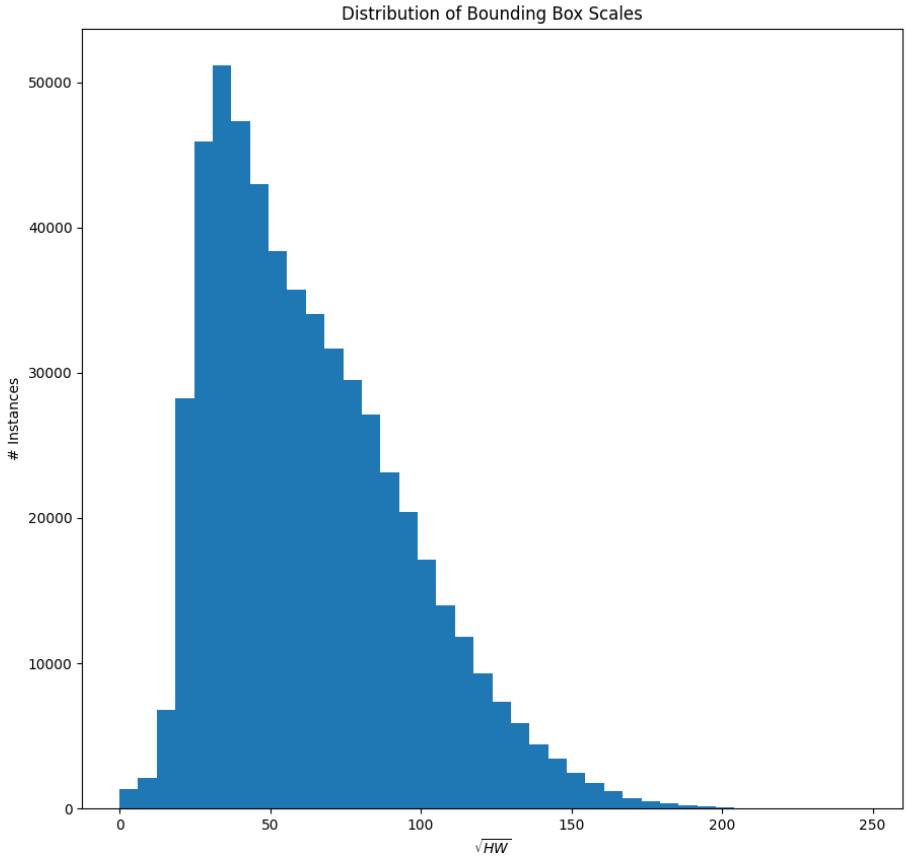


Figure 23: Distribution of bounding boxes scales $\sqrt{HW}$ for the training dataset.

RCNN anchor sampling strategy as discussed in Chapter 2. This decision was made based on performance comparisons with the FPN variant which showed worse performance when not using sampled anchors. For the sake of comparison experiments are also performed on the baseline model and baseline FPN model using both anchor sampling strategies.

4.1.2 Viewpoint Loss Correction

Local templates are chosen during training to have a pose which matches that of the target in the associated query image. A perturbation is then applied

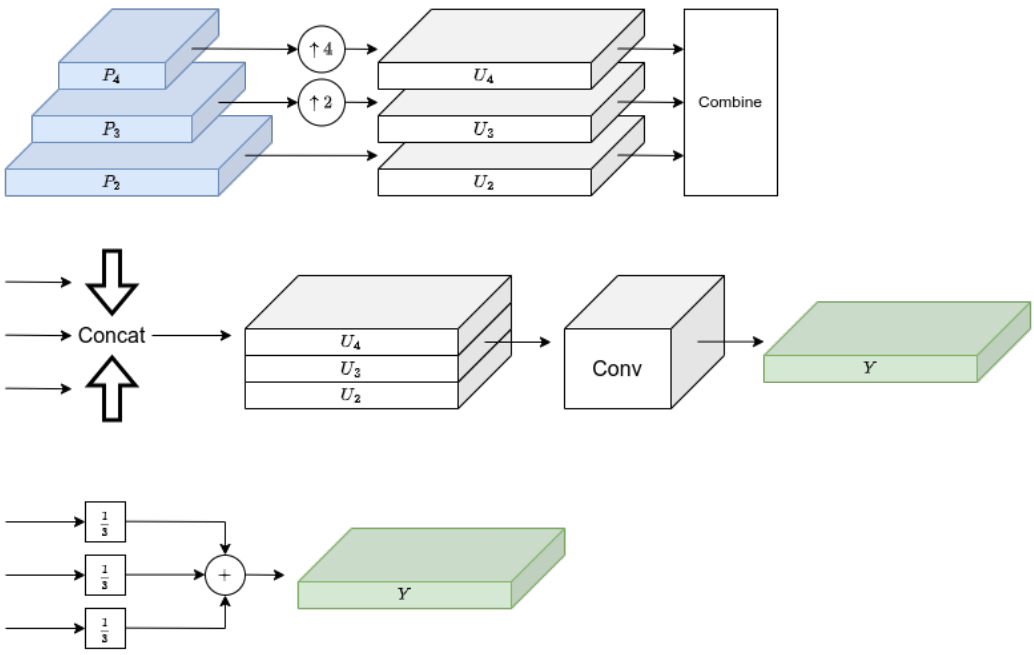


Figure 24: Diagram of pyramid level fusion via concatenation (middle) and averaging (bottom). Both methods take upsampled feature maps (top) by applying either nearest neighbor upsampling or bicubic interpolation to FPN feature maps.

to local template object poses to increase robustness to pose variation during inference. However, despite perturbations not giving an accurate correspondence between query image and template viewpoints all perturbation values are given equal weight. To compensate for variation in perturbations the loss L is weighted based on the perturbation angle θ using a cosine function.

$$\mathcal{L}_{viewpoint} = \cos(\alpha \cdot \theta) \cdot \mathcal{L} \ : \ \alpha \in [0, \frac{\pi}{2\theta_{max}}], \ \theta \in [0, \theta_{max}]$$

Where α is a hyperparameter determining how rapidly perturbation magnitude effects loss and θ_{max} is the maximum possible perturbation angle. For the query image base orientation matrix Q and perturbed orientation matrix P the perturbation angle can be calculated using the difference between rotations as follows

$$\theta = \arccos(\frac{\text{tr } PQ^\top - 1}{2})$$

Values $\alpha = 1, 2, 3$ are investigated where $\alpha = 3$ is the largest possible α for when the maximum perturbation angle is 30 degrees. An additional experiment is also performed to control for learning rate variation by using a randomly sampled θ unrelated to perturbation angle.

4.1.3 Transformers

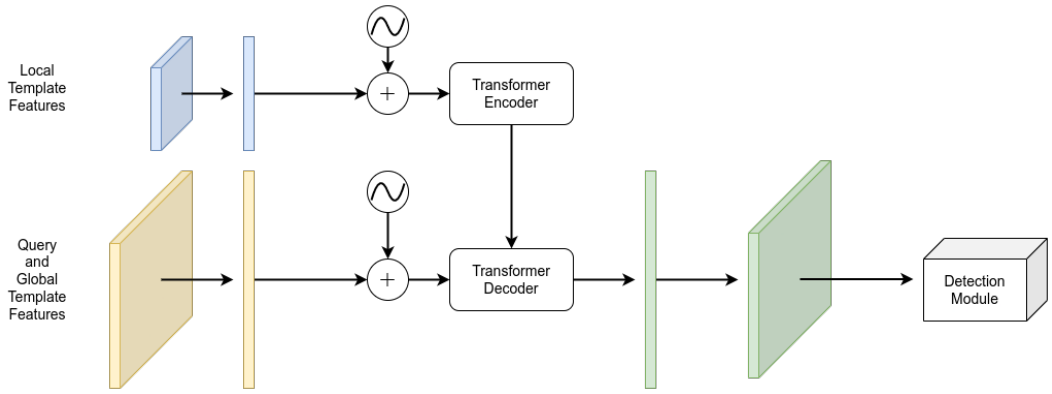


Figure 25: Usage of a vanilla transformer to aggregate local template and backbone features.

Transformers have been shown to be highly effective in the image domain. Methods such as AIT [4] and Meta-DETR [5] have further demonstrated the capability of transformers for few-shot object detection. Depth-wise cross-correlation is fundamentally limited by the fact that orientation is highly important. Viewpoint variations such as in-plane rotation may impact overall detection performance as different features maps result. This can be partially alleviated by averaging template feature maps spatially but as a result information is destroyed. In addition features may also be viewpoint specific. Depth-wise cross-correlation is a very simple comparison that essentially amounts to template-matching. Transformers have shown to be capable in domains such as image classification [6] and object

detection [7]. The usage of attention in transformers allows global comparison of sequence elements. With the global comparing capability of transformers it is possible that higher level understanding can be achieved which is less dependent on object pose. As a result it is possible that support feature poses become less important, increasing robustness to query and template pose discrepancies.

The integration of a transformer into DTOID is presented in Figure 25. A vanilla transformer is used to replace the correlation module. The pytorch [8] implementation is used specifically with default parameters. Local template features taken from the PSB are used as encoder inputs while backbone features are taken as decoder inputs. Prior to input into the transformer backbone and template feature maps are flattened into a sequence and are summed with a sinusoidal positional encoding as done in DETR[7]. The decoder produces a new sequence which is then reshaped into a new feature map with the same shape as the decoder input. After reshaping the result is passed to the detection module.

The choice of using a vanilla transformers is done primarily for implementation simplicity. Attention or transformers are not applied to the detection stage as done in other works such as DETR. Instead, transformers are used as a direct replacement for depth-wise cross-correlation to improve the aggregation of template and query image features.

4.2 Dataset Creation

In DTOID the evaluation task consists of a cluttered table containing textured objects. Both training and testing objects are from BOP Challenge [9] datasets. After contacting the authors it was found that the original dataset and dataset generation source code were lost. Instead of attempting to create a new dataset with similar parameters an alternative task is considered which involves untextured objects in a simple flat environment. Such a setting more closely mimics the

industrial environments this study focuses on.

4.2.1 Object Data

It was found that increasing the number of object classes used during training increased performance in DTOID. The Thingy10K [10] dataset contains 10,000 untextured CAD models suitable for providing a large number of untextured objects. However, the Thingy10K dataset was designed to accurately represent data on the Thingyverse platform. As a result, models come in varying file formats and may be corrupted. In addition, object scale varies largely and some objects are disjoint. Many of these issues make direct use of the Thingy10K dataset unsuitable so a filtering procedure is applied to extract usable models. First, all models over 1MB in size are removed to speed up processing time. Next, all models which are not disjoint and are in STL format are loaded into Blender 2.93 [11] using a script. A metadata file provided by the Thingy10K dataset is used to determine if objects are disjoint. Objects which failed to load into Blender are also filtered out of the selection process. Once loaded all models have planar decimation applied with default settings to remove redundant surfaces. Objects are centered about their center of mass calculated using surface area. Each model has a scale parameter s_i which is equal to the largest distance between any possible vertex pair of the object. An empirically chosen set of scaling values $e_{min} = 3cm$, $\bar{e} = 13.5cm$, and $e_{max} = 20cm$ are used to rescale and filter the remaining models. The average scale of all models $\bar{s}$ is used to scale models on all dimensions producing a new scale for each object $s'_i = s_i \frac{\bar{e}}{\bar{s}}$. Objects for which $e_{min} < s'_i < e_{max}$ are then kept. Before exporting to ply format models are decomposed into meshes containing only triangles for compatibility purposes and have red vertex coloring applied. Full Python scripts for the filtering algorithm are given in Appendix A which should be able to recreate the original set in a deterministic fashion. As a result of the filtering pro-

cess 3424 models remained. 800 are used for testing and 200 for online validation leaving 2424 for training.

4.2.2 Query Images

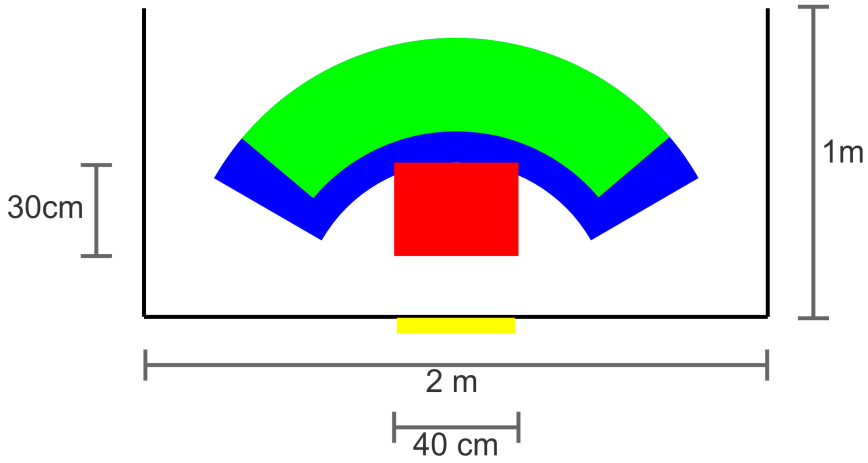


Figure 26: Cross section diagram of the query image generation scene. Red indicates the region where objects are placed, green indicates the region the camera is placed in, and blue indicates the region where the light is placed in. Note the camera sampling region is inside the light sampling region.

All query image data is created synthetically using BlenderProc 1.10.0 [12], a synthetic dataset generation automation tool based on Blender. BlenderProc provides mechanisms to describe scene creation and data sampling. Training, testing and validation splits are created using the same generation procedure, only differing by the objects used. All images are derived from one or more cameras in a generated scene. Scenes are created by placing a flat 2x2 meter gray plane at the origin surrounded by gray walls 1 meter in height to create a room with no roof. Up to 10 objects are chosen at random without replacement and placed 20cm to

50cm above the surface in a square with 40cm side length centered on the origin. Objects are given a random orientation and dropped using a physics simulation which runs until the objects have settled. Lighting is provided by a single point light placed randomly in a shell with a distance 50cm to 90cm from the origin and with a maximum angle of 60 degrees with the surface plane normal. Light colors are chosen from each color RGB channel independently at random with normalized intensity between 0.8 and 1.0 and a total energy of 10 watts. Object materials are also modified randomly, with specular and roughness values chosen randomly between 0 and 1. Camera locations are chosen in a manor similar to the point light by placing randomly in a shell 60cm to 90cm from the origin and with a maximum angle of 50 degrees with the surface plane normal. Cameras are oriented such that they point at the origin and have a random in-plane rotation. All query image cameras use a horizontal field of view of 48.45°. A cross section diagram is shown in Figure 26. All images are 480×640 in size and are rendered using CYCLES, a physics based rendering (PBR) engine, unlike [13] which uses a combination of PBR and OpenGL renders. The full BlenderProc configuration file used is given in Appendix C. In total 80,000 training images were generated with 4 images per scene, 2,000 testing images with 1 image per scene, and 200 validation images with 1 image per scene. Examples query images are given in Figure 27.

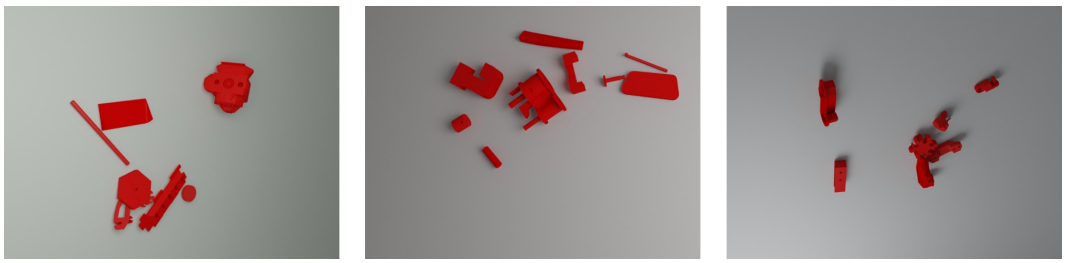


Figure 27: Example query images generated using BlenderProc.

4.2.3 Template Images

Template images are created using the BOP Toolkit [14] which provides a fast C++ based OpenGL renderer that can run without a display using OSMesa [15]. Phong shading is used for all template renders. Scenes are created by placing the object centered in front of the camera. The placement distance is chosen to be the closest distance such that the smallest sphere centered at the object's origin which could enclose the object fits in the camera frame. An additional distance is added which is randomly chosen between 0% and 10% of the object's scale for training and fixed to 5% of the object's scale for testing. Lighting is achieved through a combination of ambient lighting and a point light placed randomly on a sphere with distance 1km. Both lights produce a neutral white color. After rendering a fixed padding of 8 pixels is then added to templates resulting in a final image size of 124×124.

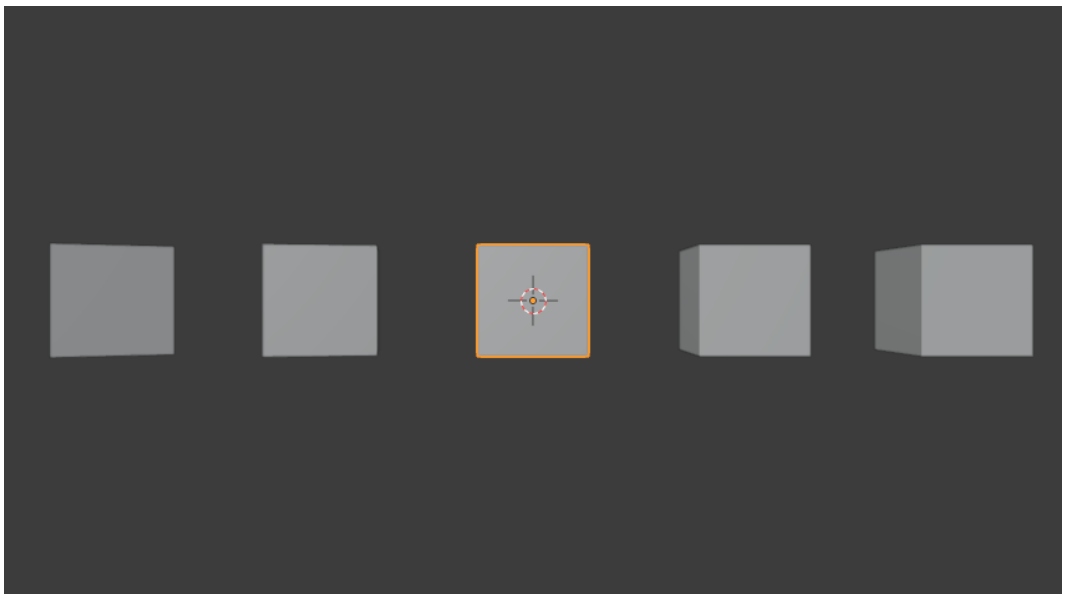


Figure 28: Visual example of the effect of perspective projection on object view. Images right of the center object have the same pose but appear increasingly different the further from the center they are. Images left of the center have been rotated to correct for this discrepancy.

During training templates are rendered online. Global template poses are

chosen randomly while local template poses are chosen based on the query image pose similar to DTOID. Local template poses are also perturbed by rotating about a random rotation vector up to 30 degrees, the optimal perturbation angle found in DTOID.

Objects were placed in front of the camera rather than at their true location to ease implementation when using the BOP Toolkit renderer. However, moving the object position results in a discrepancy between query and template images. The pose associated with a target object results in a local template image view which is more dissimilar with the query image view the further a query image object is from the image center as demonstrated in Figure 28. To counter this problem the perturbed query image pose is rotated to more closely align the template image view with the query image view. Specifically, for a given scene let R_{m2c} and t_{m2c} represent the orientation matrix and translation vector respectively that map objects from model space to the camera space. Assuming the camera points in the negative z axis, the base pose is rotated about r by angle ϕ in camera space using the following.

$$\hat{t}_{m2c} = \frac{t_{m2c}}{||t_{m2c}||}$$

$$\hat{k} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

$$r = \hat{t}_{m2c} \times \hat{k}$$

$$\phi = \arcsin(||r||)$$

During inference eight scenes are created with 20 images each sampled from the vertices of a dodecahedron. In each scene the point light location, object pose, and camera in-plane rotations are chosen randomly. As a result 160 templates are

generated for use during inference. Example template images are given in Figure 29.

Figure 29: Example template images.

List of References

[1] S. Marcel and Y. Rodriguez, "Torchvision the machine-vision package of torch," in *Proceedings of the 18th ACM International Conference on Multimedia*, ser. MM '10. New York, NY, USA: Association for Computing Machinery, 2010, p. 1485–1488. [Online]. Available: https://doi.org/10.1145/1873951.1874254

[2] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," *CoRR*, vol. abs/1502.01852, 2015. [Online]. Available: http://arxiv.org/abs/1502.01852

[3] T. Lin, P. Goyal, R. B. Girshick, K. He, and P. Dollár, "Focal loss for dense object detection," *CoRR*, vol. abs/1708.02002, 2017. [Online]. Available: http://arxiv.org/abs/1708.02002

[4] D.-J. Chen, H.-Y. Hsieh, and T.-L. Liu, "Adaptive image transformer for one-shot object detection," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2021, pp. 12 247–12 256.

[5] G. Zhang, Z. Luo, K. Cui, and S. Lu, "Meta-detr: Few-shot object detection via unified image-level meta-learning," *CoRR*, vol. abs/2103.11731, 2021. [Online]. Available: https://arxiv.org/abs/2103.11731

[6] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, "An image is worth 16x16 words: Transformers for image recognition at scale," *CoRR*, vol. abs/2010.11929, 2020. [Online]. Available: https://arxiv.org/abs/2010.11929

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

[8] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds. Curran Associates, Inc., 2019, pp. 8024–8035. [Online]. Available: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

[9] T. Hodan, F. Michel, E. Brachmann, W. Kehl, A. G. Buch, D. Kraft, B. Drost, J. Vidal, S. Ihrke, X. Zabulis, C. Sahin, F. Manhardt, F. Tombari, T. Kim, J. Matas, and C. Rother, "BOP: benchmark for 6d object pose estimation," *CoRR*, vol. abs/1808.08319, 2018.

[10] Q. Zhou and A. Jacobson, "Thingi10k: A dataset of 10,000 3d-printing models," *arXiv preprint arXiv:1605.04797*, 2016.

[11] "Blender," https://www.blender.org/download/releases/2-93/, accessed: 2022-05-07.

[12] M. Denninger, M. Sundermeyer, D. Winkelbauer, Y. Zidan, D. Olefir, M. Elbadrawy, A. Lodhi, and H. Katam, "Blenderproc," *CoRR*, vol. abs/1911.01911, 2019. [Online]. Available: http://arxiv.org/abs/1911.01911

[13] J.-P. Mercier, M. Garon, P. Giguère, and J.-F. Lalonde, "Deep template-based object instance detection," 2019. [Online]. Available: https://arxiv.org/abs/1911.11822

[14] "Bop toolkit," https://github.com/thodan/bop_toolkit, accessed: 2022-05-07.

[15] "Osmesa," https://docs.mesa3d.org/osmesa.html, accessed: 2022-05-07.

CHAPTER 5

Experiments

5.1 Experimental Setup

All experiments are performed with similar training parameters. Data augmentation is also applied to both query and template images. All numerical results are presented using the 6 COCO AP [1] metrics.

5.1.1 Training Details

Unless stated otherwise training is performed with the following parameters. The total training duration is 400,000 iterations with a batch size of 4 resulting in 20 epochs of training. AMS-Grad is used with a base learning rate of 10^{-4} and weight decay of 10^{-6} as done in DTOID [2]. The learning rate is decayed using a gamma of 0.1 at epoch 10 and epoch 15. During each epoch images are sampled uniformly at random without replacement.

Similar to DTOID, heavy data augmentation is applied. Images are randomly flipped vertically and horizontally with a probability of 0.5. Random cropping is also applied, with the result being rescaled to match the original image resolution. Cropping is achieved by selecting a random region which is fully contained within the image and has a size equal a fraction of the original image size sampled between 2/3 and 1. For query images Gaussian blur with a radius of 3 is applied with a probability of 0.2, Gaussian noise with $\sigma = 5$ is applied with probability 0.2, and brightness is altered by multiplying the image by a random factor between 0 and 2 with a probability of 0.2. In addition, the hue of both the query image and template image are changed to the same random value with probability 0.5. Although it is not investigated, augmentations with Gaussian blur, Gaussian noise, brightness and hue should enhance performance in real image scenarios and is mainly kept

Name	GPU	VRAM	RAM	CPU
dlmachine	2x 3090	48GB	128GB	AMD 3960X
aurora	1x 3090	24GB	128GB	Intel i7-10700KF
lambda	4x 2080 Ti	44GB	128GB	Intel i9-9920X

Table 2: List of machines used for running experiments.

due to being present in DTOID. All experiments were performed between the three machines presented in Table 2.

The same loss function is used as in DTOID with only loss weights adjusted. The total loss $\mathcal{L}$ can be written as a weighted sum of classification loss $\mathcal{L}_{cls}$, regression (bounding box) loss $\mathcal{L}_{reg}$, segmentation loss $\mathcal{L}_{mask}$ and center-point loss $\mathcal{L}_{center}$.

$$\mathcal{L} = \lambda_{cls}\mathcal{L}_{cls} + \lambda_{reg}\mathcal{L}_{reg} + \lambda_{mask}\mathcal{L}_{mask} + \lambda_{center}\mathcal{L}_{center}$$

Originally $\lambda_{center}, \lambda_{mask} = 20$ with other values being 1 but it was found that $\lambda_{cls} = \frac{1}{16}$ with other values being 1 gave better results and was used for all experiments. For loss function hyperparameters $\alpha = 0.25$ and $\gamma = 2$ are used for focal loss in $\mathcal{L}_{cls}$ and $\beta = 1/9$ is used for smooth L1 loss. In addition, center-point loss is adjusted slightly. In [2] center-point annotations are created using an isotropic bimodal Gaussian with a mean equal to that of the object position and standard deviation of 5. A standard deviation of 1 is used instead as a standard deviation of 5 was found to be too imprecise.

5.1.2 Evaluation Details

Results are presented in terms of the 6 COCO AP metrics as presented in Figure 30. All COCO metrics are calculated as an average across all classes. Three metrics focus on AP at specific IoUs: $AP^{IoU=0.50}$ (AP50), $AP^{IoU=0.75}$ (AP75) and $AP^{IoU=0.50:0.05:0.95}$ (AP (COCO)). While AP50 and AP75 only consider AP at IoUs of 0.5 and 0.75 respectively, AP (COCO) takes an average across all IoUs

inclusively between 0.5 and 0.95 at 0.05 increments. AP^{large} (APL), AP^{medium} (APM) and AP^{small} (APS) are similar to AP (COCO) but only consider ground truth bounding boxes which meet certain size criteria as given in Figure 30. Note that the term AP will refer to AP (COCO) throughout the rest of this chapter.

Average Precision (AP):
- AP % AP at IoU=.50:.05:.95 **(primary challenge metric)**
- $AP^{IoU=.50}$ % AP at IoU=.50 (PASCAL VOC metric)
- $AP^{IoU=.75}$ % AP at IoU=.75 (strict metric)

AP Across Scales:
- AP^{small} % AP for small objects: area < 32^2
- AP^{medium} % AP for medium objects: 32^2 < area < 96^2
- AP^{large} % AP for large objects: area > 96^2

Average Recall (AR):
- $AR^{max=1}$ % AR given 1 detection per image
- $AR^{max=10}$ % AR given 10 detections per image
- $AR^{max=100}$ % AR given 100 detections per image

AR Across Scales:
- AR^{small} % AR for small objects: area < 32^2
- AR^{medium} % AR for medium objects: 32^2 < area < 96^2
- AR^{large} % AR for large objects: area > 96^2

Figure 30: COCO Metrics. Table from [1].

To evaluate trained models only the objects in a given test image are considered. Due to high variations in performance each configuration is trained four times. Results are given by providing the max and average of the four trials across each metric. Every 10,000 training iterations the model is evaluated using the validation split and a model snapshot is taken. Graphs for validation performance for all six metrics during training can be found in the Appendix B. For each trained model testing is performed by applying the test split on the last three snapshots and taking an average of performance results. As a result trial averages are actually an average of 12 evaluations.

5.2 Experimental Results

Experiments are performed for FPNs, viewpoint loss weighting and transformers as discussed in Chapter 4. All numerical results are provided using COCO AP. Four trials are performed per experiment unless stated otherwise with maximum

and average performance provided. Image evaluation examples are provided for some experiments.

5.2.1 Feature Pyramid Networks

FPN experiments are broken up into three main sections. First, anchor configuration and anchor sampling strategies are investigated for the the baseline and FPN variant to provide a fair comparison. Next, strategies used to combine feature pyramid levels for segmentation and center-point prediction are investigated. Lastly, a larger FPN variant is trained to investigate model size. All FPN variants are trained with a base learning rate of $2.5 \cdot 10^{-5}$ which was found through a learning rate search. All other training parameters remain the same.

Anchor Sampling

As discussed in Chapter 4 three baseline models are used for comparison. DTOID is the baseline intended to replicate the original DTOID paper. DTOID-45R is trained the same as DTOID but uses the 45 anchors used in the FPN variant. Lastly DTOID-45B is the same as DTOID-45R but uses Fast RCNN anchor sampling as discussed in Chapter 2.

When comparing baseline performance for anchor sampling strategies in DTOID-45R and DTOID-45B performance does not show much of a substantial difference in terms of AP as shown in Table 3 However, looking at AP75 shows that DTOID-45R performs about 100% better on average while performing over 12 AP50 less as compared to DTOID-45B. AP50 is often considered a poor metric for object recognition performance [3] so DTOID-45R can arguably be considered a better object (instance) detector. Both DTOID-45R and DTOID-45B perform substantially better than DTOID on most metrics.

Two FPN variants are used to compare anchor sampling strategies.

DTOID+FPN-B uses the Fast RCNN anchor sampling strategy done in DTOID-45B while DTOID+FPN-R uses all anchors as done with DTOID-45R. Results in Table 3 show opposite trends in DTOID+FPN-B and DTOID+FPN-R compared to DTOID-45B and DTOID-45R. DTOID-FPN-B performs better DTOID+FPN-R on all metrics. Focal loss with full anchor sampling was originally developed for FPN detectors [4] so these results may require additional investigation. For all other FPN experiments the Fast RCNN sampling strategy is used as done in DTOID+FPN-B.

When comparing the best variants from the baseline FPNs and baseline controls, DTOID+FPN-B and DTOID-45R, a significant improvement can be seen as shown in Table 3 when using FPNs. An increase of 6.4 points is seen for AP run averages. In addition, APM and APS are increased 5.7 and 11.1 points respectively suggesting a benefit from using FPNs for small objects. However, APL is decreased 7.2 points. This can possibly be attributed to the use of lower resolution feature maps for large scale objects. AP is likely not highly impacted from this due to the lack of objects with $HW > 96^2$ as shown in Figure 23.

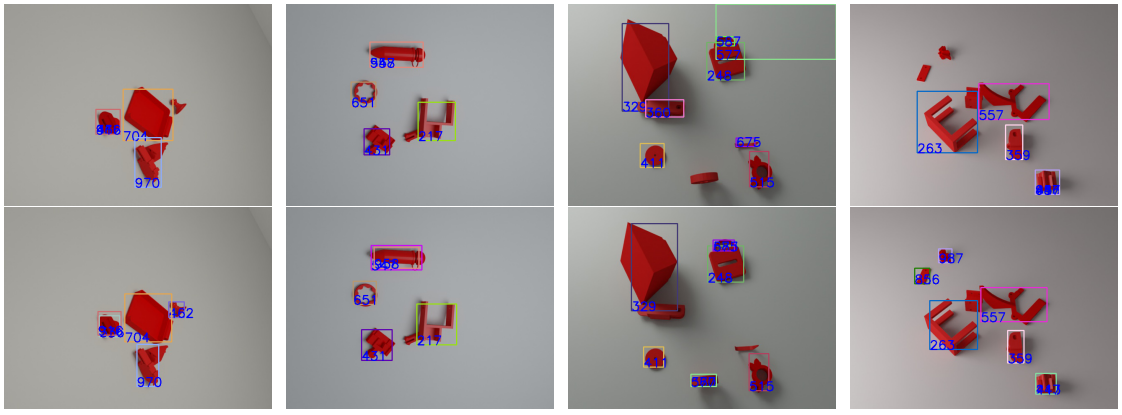


Figure 31: Visual comparison of DTOID (top) and DTOID+FPN-LARGE (bottom) variants. Images are randomly sampled. Best viewed electronically.

Pyramid Fusion

To fuse pyramid levels for segmentation and center-point prediction DTOID+FPN-B performs nearest neighbor upsampling on feature pyramid levels then averages the result. As previously discussed segmentation and center-point prediction are not used at test time and are only used to increase object instance detection performance. DTOID+FPN-BICUBIC and DTOID+FPN-CONCAT are based on DTOID+FPN-B and investigate alternative strategies to fuse pyramid levels. DTOID+FPN-BICUBIC applies bicubic upsampling and shows a significant drop in performance in terms of trial averages for all metrics as show in Table 3. It is possible that the mixing of features which results from bicubic upsampling harms localization performance. DTOID+FPN-CONCAT applies a convolution to concatenated pyramid levels rather than averaging. DTOID+FPN-CONCAT shows a possible slight increase in performance indicating that it may be best to consider pyramid levels independently.

Model Size

Lastly, DTOID+FPN-LARGE shares configuration with DTOID+FPN-B but uses the original DTOID convolution depths for local template correlation as described in Chapter 4. As shown in Table 3 DTOID+FPN-LARGE is the best performing model overall in terms of AP achieving 43.1 points, 4.6 points higher than DTOID+FPN-B and 11 points higher than DTOID-45R. These results suggest that the performance on FPN variants does not originate from over-fitting in the baseline model and that additional performance may be achievable by further increasing model size. Example inference images are shown in Figure 31 comparing DTOID and DTOID+FPN-LARGE.

Model	AP	AP_{50}	AP_{75}	AP_L	AP_M	AP_S	Size
DTOID	26.7/33.5	51.6/64.7	22.2/28.0	42.0/48.5	29.8/37.7	8.2/10.6	33M
DTOID-45R	32.1/36.9	60.4/66.7	27.7/35.3	**43.3**/**52.8**	37.3/42.1	14.5/16.1	33M
DTOID-45B	32.5/33.9	72.9/74.9	13.8/15.4	39.4/40.2	37.8/39.3	17.4/17.7	33M
DTOID+FPN-B	38.5/40.4	65.0/68.4	41.3/43.5	35.5/38.1	43.0/45.0	25.6/27.3	18M
DTOID+FPN-R	14.5/19.3	26.7/35.6	13.2/18.1	15.6/18.6	17.3/22.3	11.3/19.5	18M
DTOID+FPN-CONCAT	40.0/42.4	69.3/73.9	40.7/43.7	36.0/40.0	44.6/47.0	27.7/**31.6**	20M
DTOID+FPN-BICUBIC	32.7/40.3	55.8/69.6	34.2/41.4	28.6/37.8	36.6/45.5	22.1/26.6	18M
DTOID+FPN-LARGE	**43.1**/**45.7**	**74.5**/**78.6**	**44.0**/**47.3**	41.1/43.6	**48.2**/**51.0**	**28.4**/30.6	45M

Table 3: Comparison of baseline and FPN variants. Metrics are given in the format average/maximum.

5.2.2 Viewpoint Loss

Three models were trained with viewpoint loss weighting as discussed in Chapter 4 named DTOID+VPL-α for $\alpha = 1, 2, 3$. Comparisons with the baseline are presented in Table 4. For the three models tested DTOID+VPL-3 performed the best in terms of average performance for AP. Compared to DTOID, DTOID+VPL-3 performs better in all metrics with a 2.6 increase in AP when considering averages. However, when considering the maximum performance across the four trials tested the baseline and DTOID+VPL-2 performed the best in terms of AP. When considering run averages it appears that the use of viewpoint loss weighting provides an increase in performance.

In addition, DTOID+VPL-3-RAND displays the performance of DTOID+VPL-3 when θ is randomly sampled unrelated to perturbation angle. It is found that performance is decreased compared to DTOID and DTOID+VPL-3 suggesting variations in learning rate are not the source of increased performance in viewpoint loss variants.

When considering the three values of α used it appears that higher α values result in a higher performance when considering averages as demonstrated in Figure 32. Since $\alpha = 3$ is the maximum α value for $\theta_{max} = 30^\circ$ it is advisable to always set α to its maximum configurable value $\alpha = \frac{\pi}{2\theta_{max}}$.

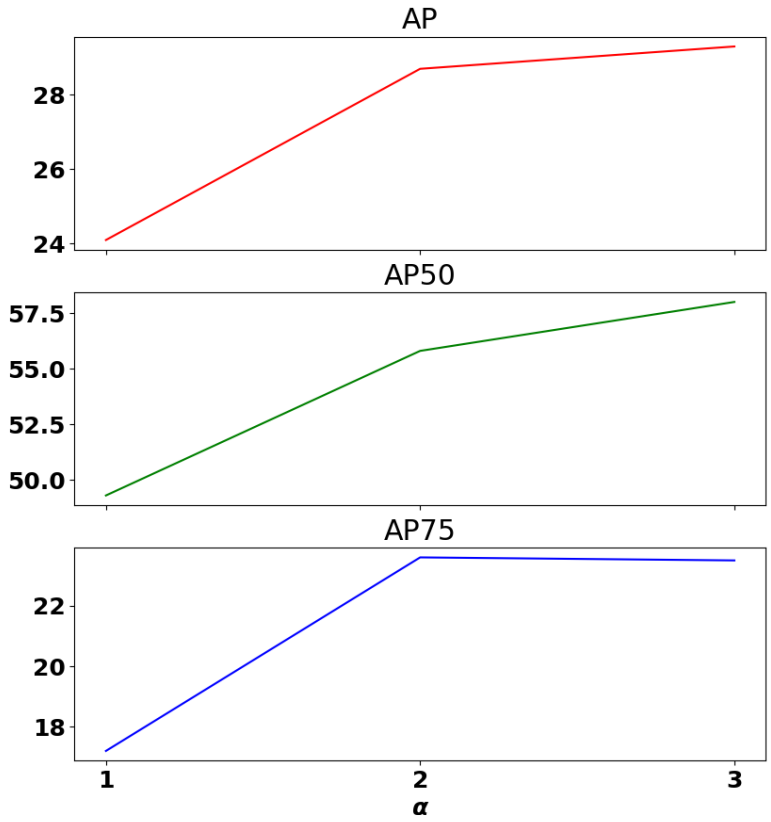


Figure 32: AP, AP50, and AP75 for run averages with respect to α.

5.2.3 Transformers

A transformer is integrated into DTOID as described in Chapter 4. Due to computational and time constraints only a single trial is performed. In addition, an alternative training configuration is used. A batch size of 8 is used with a base learning rate of $2 \cdot 10^{-5}$ found through a learning rate search. 10x more training is performed as in [5] for a total of 2,000,000 iterations, or 200 epochs. Learning rate is inspired by AIT [6] and is computed as follows during training

$$\text{lr} = \text{base\_lr} \cdot \min(\text{steps}/\text{warm\_steps}, (gamma)^{\lfloor steps/\text{step\_period} \rfloor})$$

Model	AP	AP_{50}	AP_{75}	AP_L	AP_M	AP_S
DTOID	26.7/**33.5**	51.6/64.7	22.2/28.0	42.0/48.5	29.8/**37.7**	8.2/10.6
DTOID+VPL-1	24.1/28.9	49.3/58.5	17.2/21.6	35.4/41.6	27.5/32.4	7.8/10.7
DTOID+VPL-2	28.7/33.4	55.8/**66.8**	**23.6**/25.8	**45.0**/**52.2**	31.9/36.5	8.9/**11.3**
DTOID+VPL-3	**29.3**/32.7	**58.0**/62.6	23.5/**28.6**	42.2/46.4	**33.2**/36.5	**9.6**/10.7
DTOID+VPL-3-RAND	24.8/28.7	49.6/56.4	18.9/22.5	39.8/46.1	28.2/32.5	7.0/9.1

Table 4: Comparison of baseline and viewpoint loss variants. Metrics are given in the format average/maximum.

Model	AP	AP_{50}	AP_{75}	AP_L	AP_M	AP_S
DTOID	26.7/33.5	51.6/64.7	22.2/28.0	42.0/48.5	29.8/37.7	8.2/10.6
DTOID+TF	20.2	41.7	15.8	34.7	22.7	5.6

Table 5: Comparison of baseline and transformer variant.

Where warm_steps is set to 20,000 and step_period is set to 80,000. Learning rate grows linearly up to the base learning rate for the first 20,000 iterations, followed by a learning rate decay by a gamma of 0.9 every 80,000 iterations. Validation is done every 20,000 iterations rather than every 10,000 iterations as done in other experiments. Snapshots are still taken every 10,000 iterations for testing.

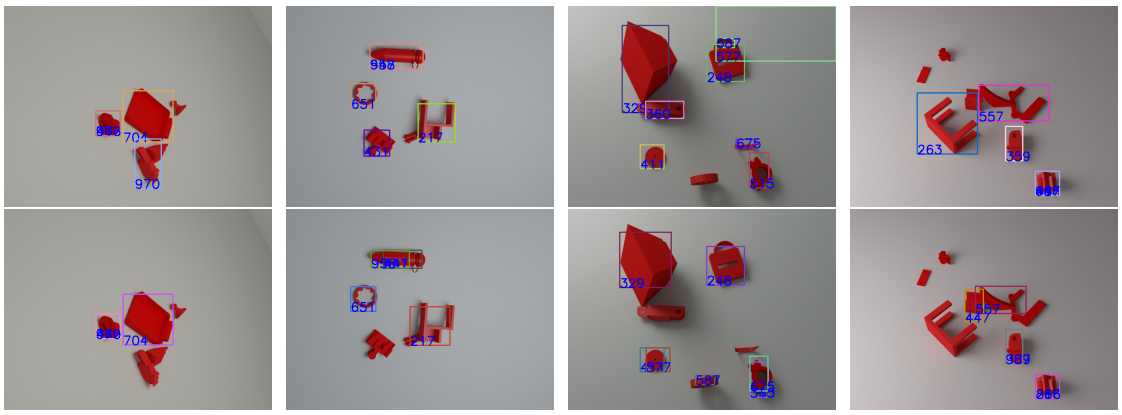


Figure 33: Visual comparison of DTOID (top) and DTOID+TF (bottom) variants. Images are randomly sampled. Best viewed electronically.

Numerical and visual comparisons with DTOID can be seen in Table 5 and Figure 33 with DTOID+TF indicating the transformer variant. While DTOID+TF was successfully trained to perform object instance detection its overall performance is substantially less than that of the baseline, DTOID. Validation per-

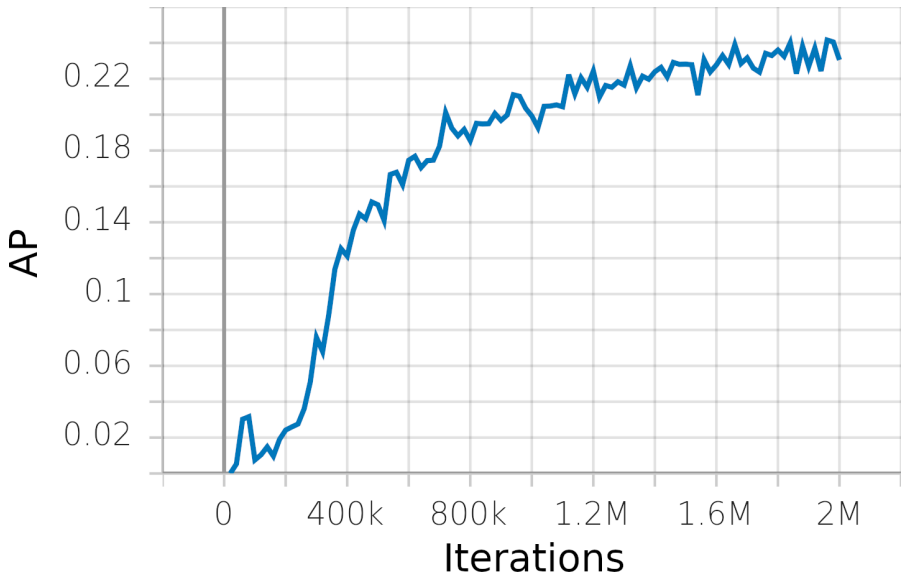


Figure 34: Validation performance in AP during training for DTOID+TF. Additional data can be found in Appendix B.

formance during training is shown in Figure 34. Performance appears to have plateaued suggesting the model was sufficiently trained. Visual results shown in Figure 33 show no strange or unusual failure conditions. Many incorrect predictions in DTOID+TF appear to be associated with the wrong object which also often happens in DTOID.

One notable difference from DTOID+TF and other transformer models is that the inputs for the encoder and decoder can be considered to be reversed. Typically the encoder takes the sequence to be *transformed*. For example, vanilla transformers performing machine translation will take the source language as the encoder input. For vision both DETR [5] and ViT [7] take images as encoder inputs. We instead use image data input at the decoder with the intent to transform it for object detection using template features. While this allows a feature map of the same size as the input to be created it is separated from the general philosophy typically used for transformers and may be the cause of worsened performance. As a result future work may benefit from taking an alternative approach based on

this observation.

List of References

[1] "Coco metrics," https://cocodataset.org, accessed: 2022-06-23.

[2] J.-P. Mercier, M. Garon, P. Giguère, and J.-F. Lalonde, "Deep template-based object instance detection," 2019. [Online]. Available: https://arxiv.org/abs/1911.11822

[3] J. H. Hosang, R. Benenson, P. Dollár, and B. Schiele, "What makes for effective detection proposals?" *CoRR*, vol. abs/1502.05082, 2015.

[4] T. Lin, P. Goyal, R. B. Girshick, K. He, and P. Dollár, "Focal loss for dense object detection," *CoRR*, vol. abs/1708.02002, 2017. [Online]. Available: http://arxiv.org/abs/1708.02002

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

[6] D.-J. Chen, H.-Y. Hsieh, and T.-L. Liu, "Adaptive image transformer for one-shot object detection," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2021, pp. 12 247–12 256.

[7] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, "An image is worth 16x16 words: Transformers for image recognition at scale," *CoRR*, vol. abs/2010.11929, 2020. [Online]. Available: https://arxiv.org/abs/2010.11929

CHAPTER 6

Conclusion

This study investigates improving the object instance detection performance of DTOID, a zero-shot meta-learning object instance detection model. Most few-shot object detection models use fine-tuning, or in some cases meta-learning, to learn from few labeled examples. By comparison, DTOID uses 3D models of objects to create controlled renders in place of labeled data. However, DTOID's object detection performance has a significant room for improvement providing a reason for this study.

Experiments are conducted on untextured components in a simple setting intended to mimic objects manufactured industrial settings. Objects produced in industrial settings often have CAD models readily available making DTOID particularly useful in such a scenario. Both training and testing data derive from over 3000 3D models extracted from the Thingi10k dataset [1]. 80,000 images were generated for training using physics based rendering in Blender. In addition, fast template rendering allowed online generation of template examples for training.

Three alterations to DTOID were proposed and investigated with a primary research question of if they can improve object instance detection performance. Six numerical performance metrics from the COCO [2] benchmark are used to compare performance. In addition, visual examples are provided. As a result of this study three alterations to DTOID are contributed in addition to experimental results demonstrating the object instance detection capability of those methods.

The first alteration investigated is the addition of a feature pyramid network (FPN) to DTOID's backbone. FPNs are highly researched and are commonly used to enhance performance on small objects in object detection. Several experiments

are performed using FPNs controlling for factors such as anchors used, model parameter size and training method. Results show an increase in performance independent of the anchors used. In addition, a large parameter FPN-based DTOID variant is able to achieve an over 60% performance gain compared to the baseline model.

The second alteration investigated involves a novel loss weighting strategy During training local template images contain objects which are perturbed with respect to the query image to increase robustness at test time. However, these perturbed images are given equal weight during training compared to ones with high correspondence to query image data. A novel viewpoint loss weighting strategy is proposed to give more value to examples with less perturbation. By employing this strategy a modest increase in object instance detection performance of 2.5 AP is found when considering trial averages. In addition, the hyperparameter α used by viewpoint loss weighting is found to have a positive correlation with object instance detection performance. Lastly, an experiment is performed which applies viewpoint loss weighting with a randomly sampled perturbation value unrelated to the one actually used. When using a unrelated perturbation value for viewpoint loss weighting it is found that performance decreases suggesting that VPL increases performance as result of viewpoint correspondence and not as a result of learning rate change.

The third alteration investigated is the usage of a transformer to aggregate query and template features. Transformers have received significant attention in recent year in a variety of tasks including computer vision. Different hyperparameters are used to better suite the transformer variant of DTOID. Despite this, performance is unable to meet that of the baseline. Possible causes are discussed, including the difference in how transformer inputs are treated in this work com-

pared to others.

In the future transformers may be of further interest for investigation due to the power they have shown in other works. Rather than integrating transformers into the current DTOID architecture it may be more effective to utilize existing transformer based object detectors such as DETR [3]. DETR using image features as encoder inputs and a set of learned embeddings called object queries for decoder inputs. Object queries are what ultimately represent detected objects in DETR's output. One possible way to incorporate templates into DETR is by transforming template features into object queries. Lastly, transformers may allow the usage of multiple templates for PSB input to allow better 3D understanding of objects. In addition, multi-template input may provide an advantage over fine-tuned and other meta-learning methods since significant computation may be shared. Ultimately it may be possible to encode 3D models into a single feature directly rather than rendering them, removing the need to have images of target objects completely.

List of References

[1] Q. Zhou and A. Jacobson, "Thingi10k: A dataset of 10,000 3d-printing models," *arXiv preprint arXiv:1605.04797*, 2016.

[2] "Coco metrics," https://cocodataset.org, accessed: 2022-06-23.

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

APPENDIX A

Blender Scripts

A.1 Main Script

```
import logging
import numpy as np
import shutil
import subprocess
import sys
import os
import glob
import yaml
import tarfile
import json
import requests
import click
import random
import gdown
import datetime
from tqdm import tqdm
from pathlib import Path

project_dir = Path(__file__).resolve().parents[1]
sys.path.append(str(project_dir))

import src.config as config

sys.path.append(config.BOP_TOOLKIT_PATH)

from bop_toolkit_lib import inout
from bop_toolkit_lib import misc


ldr_tmp = '/tmp/dtoid_model.ldr'
obj_tmp = '/tmp/dtoid_model.obj'
output_dir = 'data/raw/models'
output_tfile = os.path.join(output_dir, 'obj_{:06d}.ply')
split_file = 'data/raw/bop_data/extemps_query/split_obj_ids.yaml'


# vertex color to apply to all objects
color = ["255", "0", "0"]


def md5_file(f):
    return subprocess.check_output("md5sum {}".format(f)).strip()


def load_obj(f):
    verts = []
    norms = []
    faces = []
    with open(f, 'r+') as m:
        for row in m:
            line = row[:-1].split(' ')
            if line[0] == 'v':
                verts.append(np.array(line[1:]).astype(np.float))
            elif line[0] == 'vn':
                norms.append(line[1:])
            elif line[0] == 'f':
                # ignore texture coordinates
                if any('//' in v for v in line):
                    line = [v.split('//')[0] for v in line]
                face = (np.array(line[1:]).astype(int) - 1).astype(str).tolist()
                faces.append(face)
    return verts, norms, faces


def export_dataset(split_data):

    logger = logging.getLogger(__name__)

    # save split file object ids
    with open(split_file, 'w') as s:
        yaml.dump(split_data, s, sort_keys=False)

    models_info = {}
    for model_tpath in glob.glob(os.path.join(output_dir, '*.ply')):

        obj_id = int(os.path.split(model_tpath)[1].split('.')[0].split('_')[1])

        logger.info('Processing model of object {}...'.format(obj_id))
```

```
79
80          model = inout.load_ply(model_tpath)
81
82          # Calculate 3D bounding box.
83          ref_pt = np.array(list(map(float, model['pts'].min(axis=0).flatten())))
84          size = np.array(list(map(float, (model['pts'].max(axis=0) - ref_pt).flatten())))
85
86          # Calculated diameter.
87          diameter = misc.calc_pts_diameter(model['pts'])
88
89          models_info[obj_id] = {
90              'min_x': ref_pt[0], 'min_y': ref_pt[1], 'min_z': ref_pt[2],
91              'size_x': size[0], 'size_y': size[1], 'size_z': size[2],
92              'diameter': diameter
93          }
94
95      # Save the calculated info about the object models.
96      models_info_path = os.path.join(output_dir, 'models_info.json')
97      inout.save_json(models_info_path, models_info)
98
99
100 @click.command()
101 @click.argument('dataset', type=click.Choice(['thingyverse', 'leocad']))
102 def main(dataset):
103     """ Pulls models given in manifest and processes them for usage
104     """
105     logger = logging.getLogger(__name__)
106     logger.info('Starting...')
107
108     if not os.path.isdir(output_dir):
109         os.makedirs(output_dir)
110     else:
111         # remove old files
112         for f in glob.glob(os.path.join(output_dir, '*')):
113             os.remove(f)
114
115     if dataset == 'thingyverse':
116         dataset_dir_raw = 'scripts/thingyverse_raw'
117         dataset_dir_raw_loc = '{}/Thingi10K/raw_meshes'.format(dataset_dir_raw)
118         dataset_dir = 'scripts/thingyverse'
119         dataset_dir_loc = '{}/Thingi10K/raw_meshes'.format(dataset_dir)
120         out_file_name = 'Thingi10K.tar.gz'
121         dataset_file = 'scripts/{}'.format(out_file_name)
122         src_url = 'https://drive.google.com/u/0/uc?id=0B4_KyPW4T9oGRHdMTGZnVDFHLUU'
123         mapping_file = 'scripts/thingyverse_mapping_output.yaml'
124
125         archive_sum = 'caf1c2e65c97aea8618a2431506c04c5'
126         manifest_file = 'scripts/thingyverse_manifest.yaml'
127         # TODO add integrity checks
128         if not os.path.isdir(dataset_dir):
129             logger.info("Missing processed models, checking raw models")
130             # TODO add integrity checks
131             if not os.path.isdir(dataset_dir_raw):
132                 logger.info("Missing raw models, checking compressed models")
133                 # TODO add integrity checks
134                 if not os.path.isfile(dataset_file):
135                     logger.info("Missing compressed models, starting download...")
136                     gdown.download(src_url, dataset_file, quiet=False)
137                 with tarfile.open(dataset_file) as t:
138                     t.extractall(dataset_dir_raw)
139                 logger.info("Finished extracting")
140                 total_dropped = 0
141                 for stl_file in glob.glob(os.path.join(dataset_dir_raw_loc, "*.stl")):
142                     # drops files larger than 1MB
143                     size = os.path.getsize(stl_file)
144                     if size >= 1e6:
145                         os.remove(stl_file)
146                         total_dropped += size
147                 logging.info("Removed {}| MB".format(total_dropped // 1e6))
148             p = subprocess.Popen("blender -b -P scripts/process_models.py", shell=True)
149             p.wait()
150             if p.returncode != 0:
151                 raise Exception("Blender failed to process models!")
152
153         # export models to ply
154         with open(manifest_file, 'r') as f:
155             manifest = yaml.safe_load(f)
156             mapping = {}
157             mapping['created'] = str(datetime.datetime.now(datetime.timezone.utc))
158             idx = 1
159             split_obj_ids = {}
160             ply_models = glob.glob(os.path.join(dataset_dir_loc, "*.ply"))
161             key_fn = lambda x: int(os.path.splitext(os.path.split(x)[1])[0])
162             ply_models = sorted(ply_models, key=key_fn)
163             # handle test and val which have fixed size
164             for split, counts in manifest.items():
165                 mapping[split] = {}
166                 obj_ids = []
167                 if counts <= 0:
168                     counts = len(ply_models)
```

```
                for i in range(counts):
                    ply = ply_models.pop(0)
                    output_file = output_tfile.format(idx)
                    logging.info("{} is {} of {}".format(ply, output_file, split))
                    mapping[split][output_file] = ply
                    shutil.copyfile(ply, output_file)
                    obj_ids.append(idx)
                    idx += 1
                split_obj_ids[split] = obj_ids

            inout.save_json(mapping_file, mapping)

            export_dataset(split_obj_ids)

    elif dataset == 'leocad':
        manifest_file = 'scripts/leocad_manifest.yaml'

        # export models to ply
        with open(manifest_file, 'r') as f:
            manifest = yaml.safe_load(f)
            idx = 1
            split_obj_ids = {}
            for split, model_names in manifest.items():
                obj_ids = []
                for model_name in model_names:
                    logger.info('{} : {}'.format(idx, model_name))
                    # need to convert leocad format for their models
                    # create ldr for model
                    with open(ldr_tmp, 'w') as m:
                        m.write('1 4 0 0 0 1 0 0 0 1 0 0 0 1 {}'.format(model_name))
                    # export to obj
                    p = subprocess.Popen(
                        'leocad -obj {} {}'.format(obj_tmp, ldr_tmp), shell=True)
                    p.wait()
                    if p.returncode != 0:
                        exit(p.returncode)
                    # convert obj to ply
                    verts, norms, faces = load_obj(obj_tmp)

                    # geometrically center the object
                    verts_nd = np.stack(verts)
                    geo_mean = verts_nd.mean(0)
                    verts_nd -= geo_mean
                    verts = verts_nd.astype(str).tolist()

                    # ply export
                    with open(output_tfile.format(idx), 'w') as m:
                        m.write("ply\n")
                        m.write("format ascii 1.0\n")
                        m.write("element vertex {}\n".format(len(verts)))
                        m.write("property float x\n")
                        m.write("property float y\n")
                        m.write("property float z\n")
                        m.write("property float nx\n")
                        m.write("property float ny\n")
                        m.write("property float nz\n")
                        m.write("property uchar red\n")
                        m.write("property uchar green\n")
                        m.write("property uchar blue\n")
                        m.write("element face {}\n".format(len(faces)))
                        m.write("property list uchar uint vertex_indices\n")
                        m.write("end_header\n")
                        for vert, norm in zip(verts, norms):
                            vertstr = " ".join(
                                    vert +
                                    norm +
                                    color) + "\n"
                            m.write(vertstr)
                        for face in faces:
                            line = [str(len(face))] + face
                            m.write(" ".join(line) + "\n")

                    obj_ids.append(idx)
                    idx += 1

                split_obj_ids[split] = obj_ids

            export_dataset(split_obj_ids)

if __name__ == '__main__':
    log_fmt = '%(asctime)s - %(name)s - %(levelname)s - %(message)s'
    logging.basicConfig(level=logging.INFO, format=log_fmt)

    main()
```

A.2 Blender Processing Script

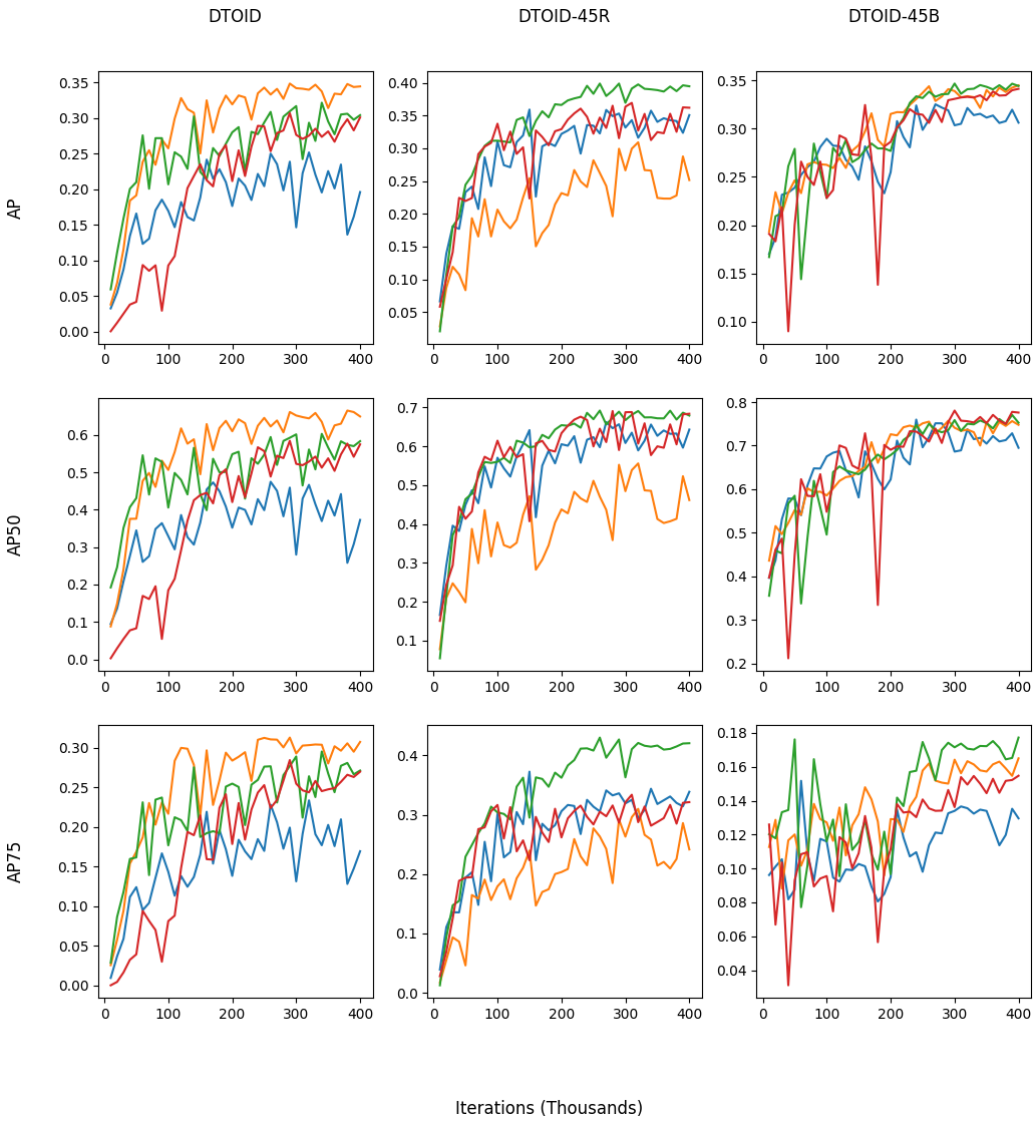
```
import bpy
import numpy as np
import math
import os
import csv
import glob
import shutil
import logging
import sys
from scipy.spatial import distance

input_dir = './scripts/thingyverse_raw/Thingi10K/raw_meshes'
output_dir = './scripts/thingyverse/Thingi10K/raw_meshes'
meta_file = './scripts/thingyverse_meta.csv'

log_fmt = '%(asctime)s - %(name)s - %(levelname)s - %(message)s'
logging.basicConfig(level=logging.INFO, format=log_fmt)
logger = logging.getLogger(__name__)

min_size = 30
target_scale = 135
max_size = 200

if os.path.isdir(output_dir):
    shutil.rmtree(output_dir)
os.makedirs(output_dir)

model_info = {}
with open(meta_file) as f:
    reader = csv.DictReader(f)
    for row in reader:
        model_info[row['ID']] = row


non_single = 0
failed = 0
files = glob.glob(os.path.join(input_dir, "*.stl"))
logger.info("{} models found".format(len(files)))
for i, stl_file in enumerate(files):
    print("{}/{}".format(i, len(files)))

    file_name = os.path.split(stl_file)[1]
    name = os.path.splitext(file_name)[0]
    output_file = os.path.join(output_dir, name + ".ply")

    if name not in model_info or model_info[name]['Single Component'] == 'FALSE':
        non_single += 1
        continue

    try:
        bpy.ops.import_mesh.stl(filepath=stl_file)
        obj = bpy.context.selected_objects[0]

        scene = bpy.context.scene

        # decimate to remove redundant faces
        obj.modifiers.new('decimator', 'DECIMATE')
        obj.modifiers.values()[0].decimate_type = 'DISSOLVE'

        # center to center of mass
        bpy.ops.object.origin_set(
            type='ORIGIN_CENTER_OF_MASS', center='MEDIAN')

        if len(obj.to_mesh().vertices.items()) == 0:
            raise Exception("empty mesh")

        # center
        obj.location = [0, 0, 0]

        bpy.ops.export_mesh.ply(filepath=output_file,
                                use_selection=True, use_mesh_modifiers=True)

        # clean up
        bpy.ops.object.delete()
    except:
        # for various reasons some models fail to load, we just drop them
        if os.path.isfile(output_file):
            os.remove(output_file)
        failed += 1

logger.info("{} removed due to consisting of multiple parts".format(non_single))
logger.info("{} removed due to unknown loading error".format(failed))

sizes = {}

files = glob.glob(os.path.join(output_dir, "*.ply"))
logger.info("{} remain".format(len(files)))
logger.info("getting object sizes")
for i, ply_file in enumerate(files):
```

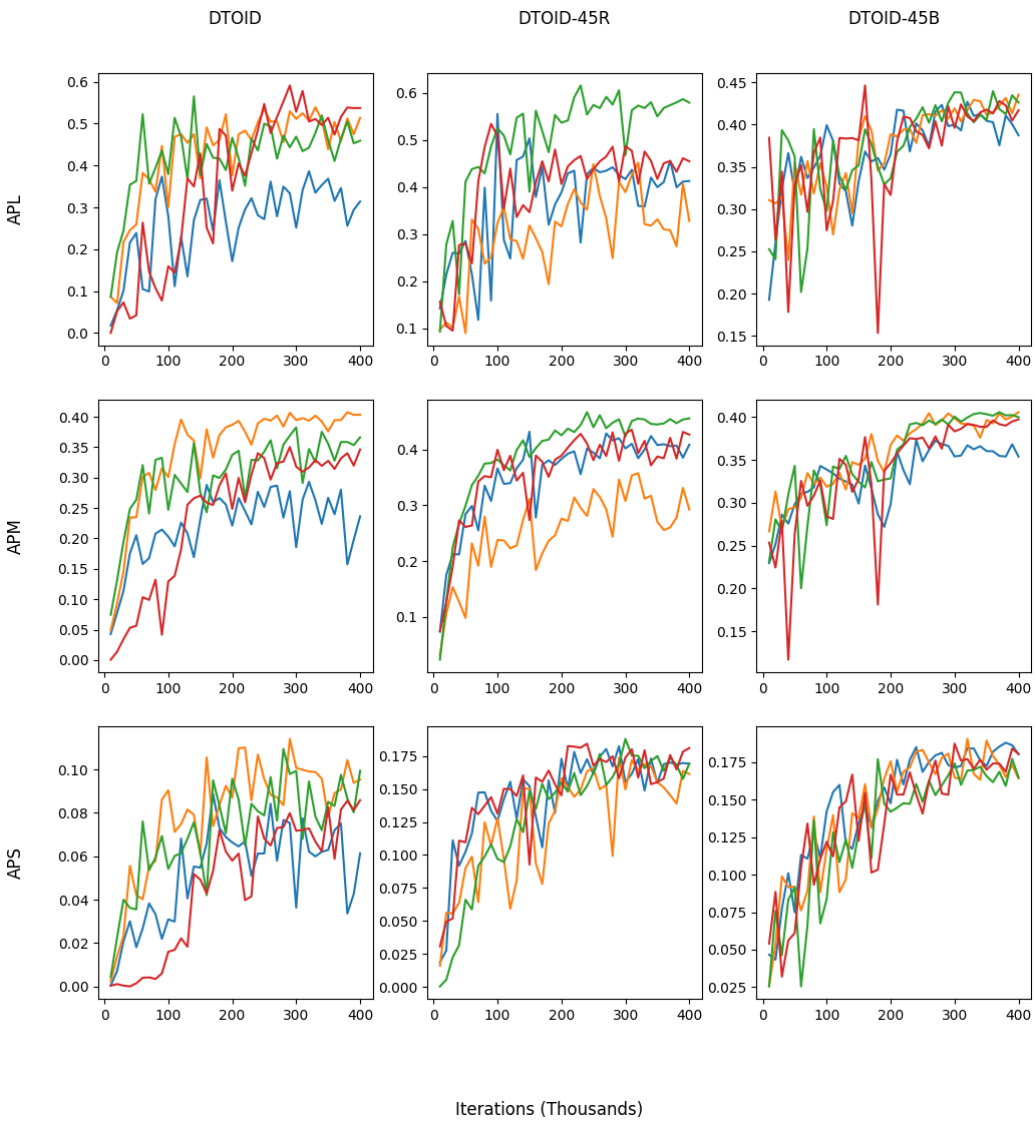
```
90          print("{}/{}".format(i, len(files)))
91          bpy.ops.import_mesh.ply(filepath=ply_file)
92          obj = bpy.context.selected_objects[0]
93
94          size = len(obj.to_mesh().vertices.values())
95          verts = np.empty(size*3, dtype=np.float64)
96          obj.to_mesh().vertices.foreach_get('co', verts)
97          verts.shape = (size, 3)
98
99          dists = distance.cdist(verts, verts, 'euclidean')
100         diameter = np.max(dists)
101
102         sizes[ply_file] = diameter
103
104         bpy.ops.object.delete()
105
106     scale_factor = target_scale / np.mean(list(sizes.values()))
107     logger.info("pre-filter size: {}".format(len(sizes)))
108     scaled_sizes = dict(map(lambda x: (x[0], x[1] * scale_factor), sizes.items()))
109     filtered_sizes = dict(filter(lambda x: x[1] > min_size and x[1] < max_size, scaled_sizes.items()))
110     logger.info("post-filter size: {}".format(len(filtered_sizes)))
111
112     logger.info("scaling and filtering object files")
113     files = glob.glob(os.path.join(output_dir, "*.ply"))
114     for i, ply_file in enumerate(files):
115         print("{}/{}".format(i, len(files)))
116         if ply_file not in filtered_sizes.keys():
117             os.remove(ply_file)
118         else:
119             bpy.ops.import_mesh.ply(filepath=ply_file)
120             bpy.ops.transform.resize(value=[scale_factor]*3)
121
122             # convert to triangles to be compatiable with bop toolkit
123             bpy.ops.object.mode_set(mode='EDIT')
124             bpy.ops.mesh.select_all(action='SELECT')
125             bpy.ops.mesh.quads_convert_to_tris()
126
127             # set vertex colors to red, similar to leocad
128             bpy.ops.object.mode_set(mode='VERTEX_PAINT')
129             mesh = bpy.context.active_object.data
130
131             for polygon in mesh.polygons:
132                 for i, index in enumerate(polygon.vertices):
133                     loop_index = polygon.loop_indices[i]
134                     mesh.vertex_colors.active.data[loop_index].color = [1, 0, 0, 1]
135             bpy.ops.object.mode_set(mode='OBJECT')
136
137             bpy.ops.export_mesh.ply(filepath=ply_file,
138                                     use_selection=True, use_mesh_modifiers=True, use_ascii=True, use_colors=True)
139             bpy.ops.object.delete()
140
141     files = glob.glob(os.path.join(output_dir, "*.ply"))
142     logger.info("{} remain".format(len(files)))
143     logger.info("min: {}, mean: {}, max: {}".format(min(filtered_sizes.values()), np.mean(list(filtered_sizes.values())), max(filtered_sizes.values())))
```

APPENDIX B

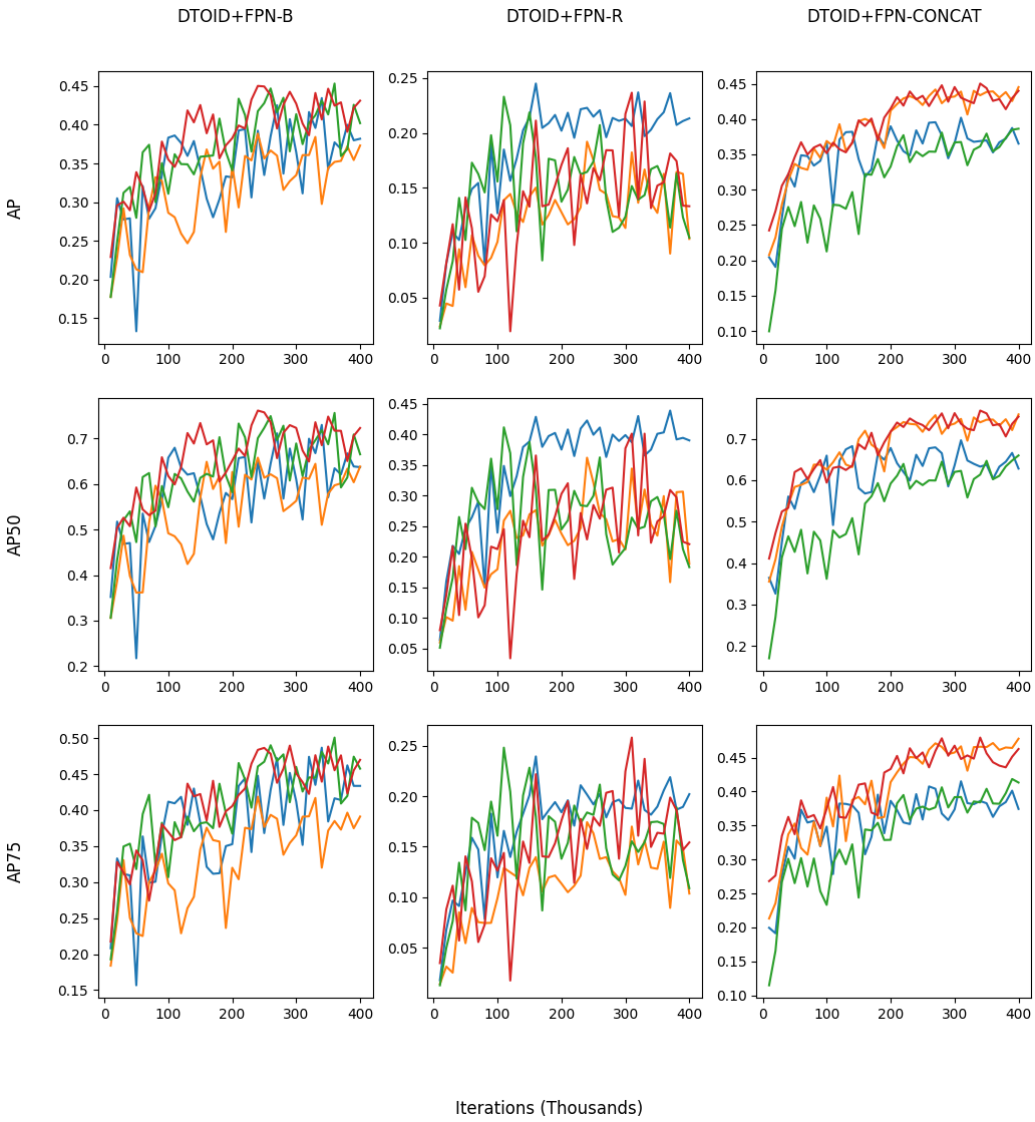
Validation Performance

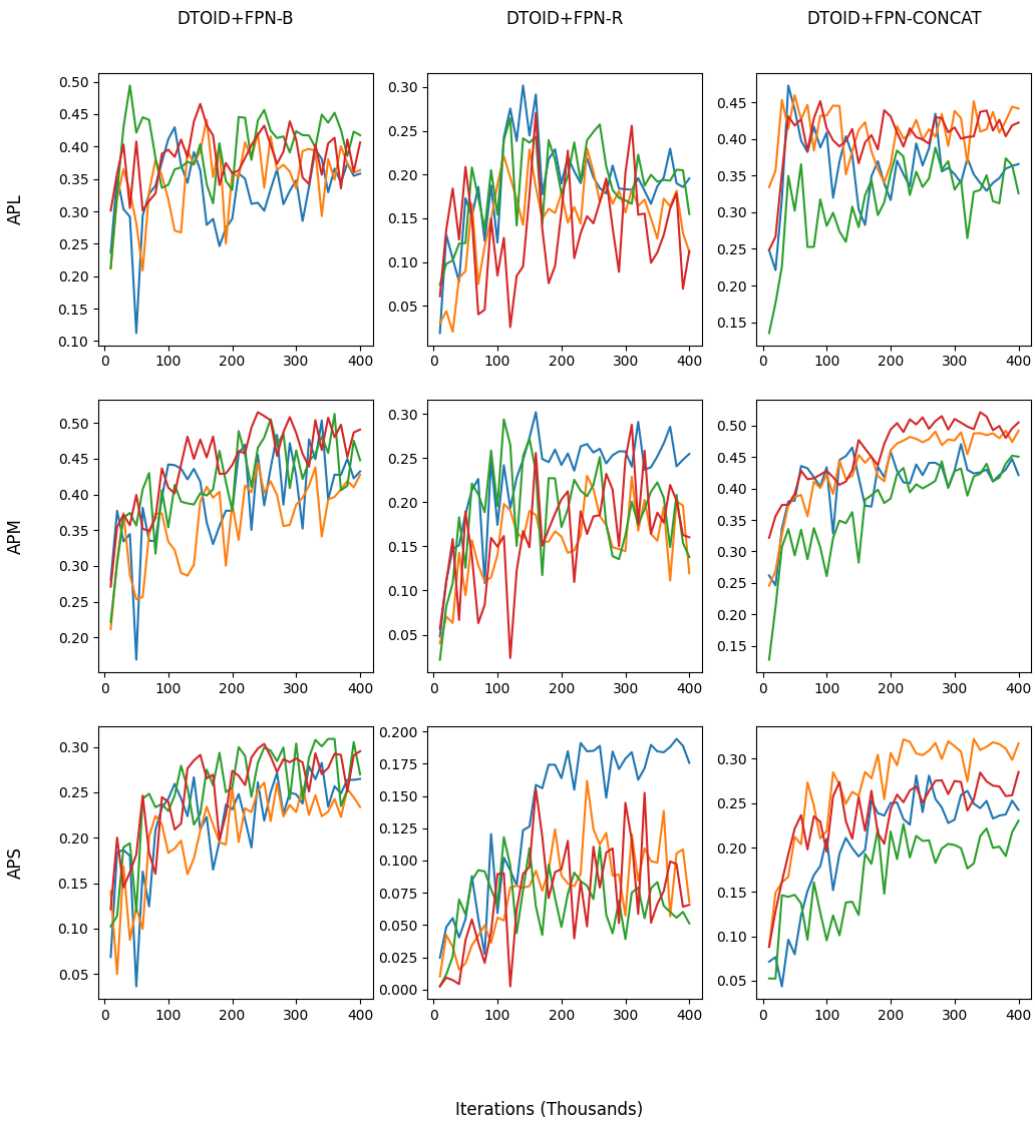
B.1 DTOID

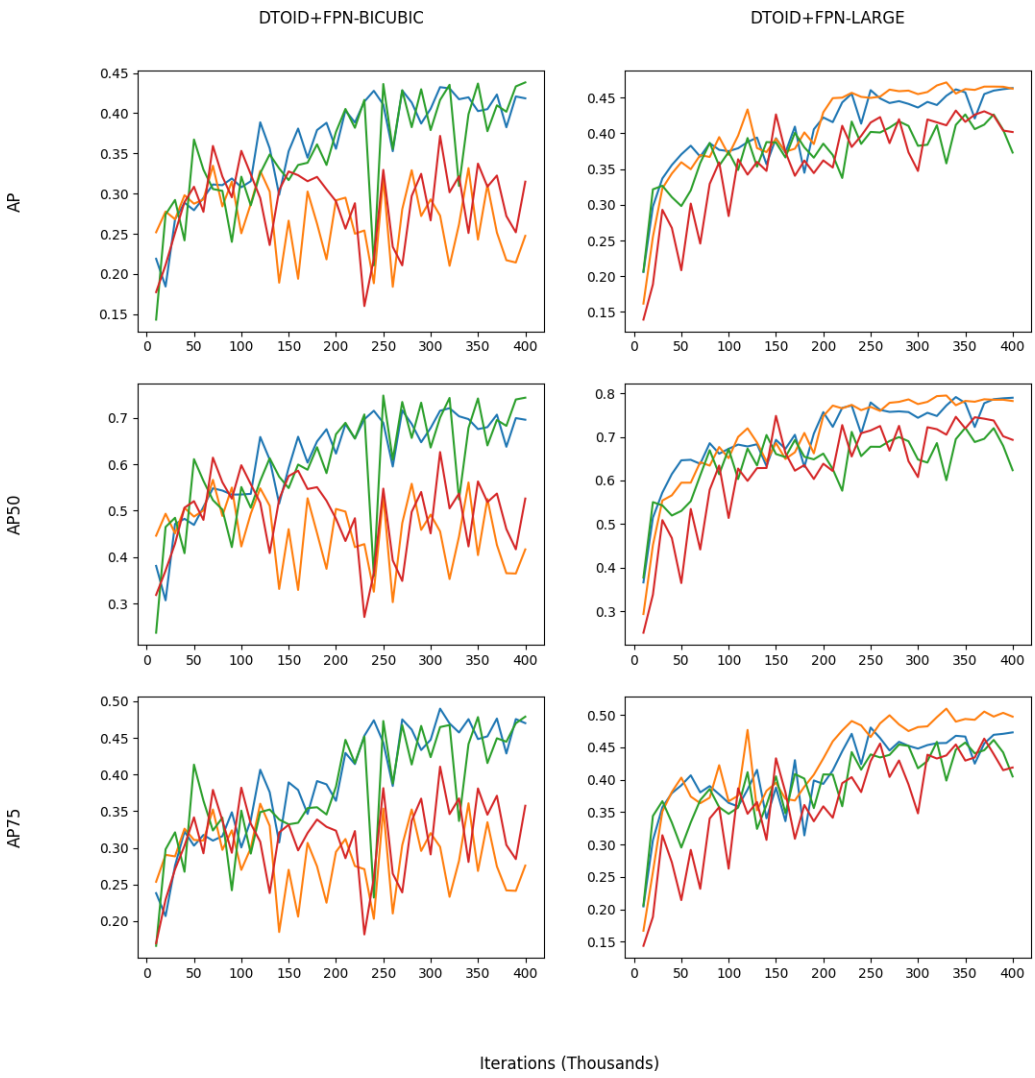


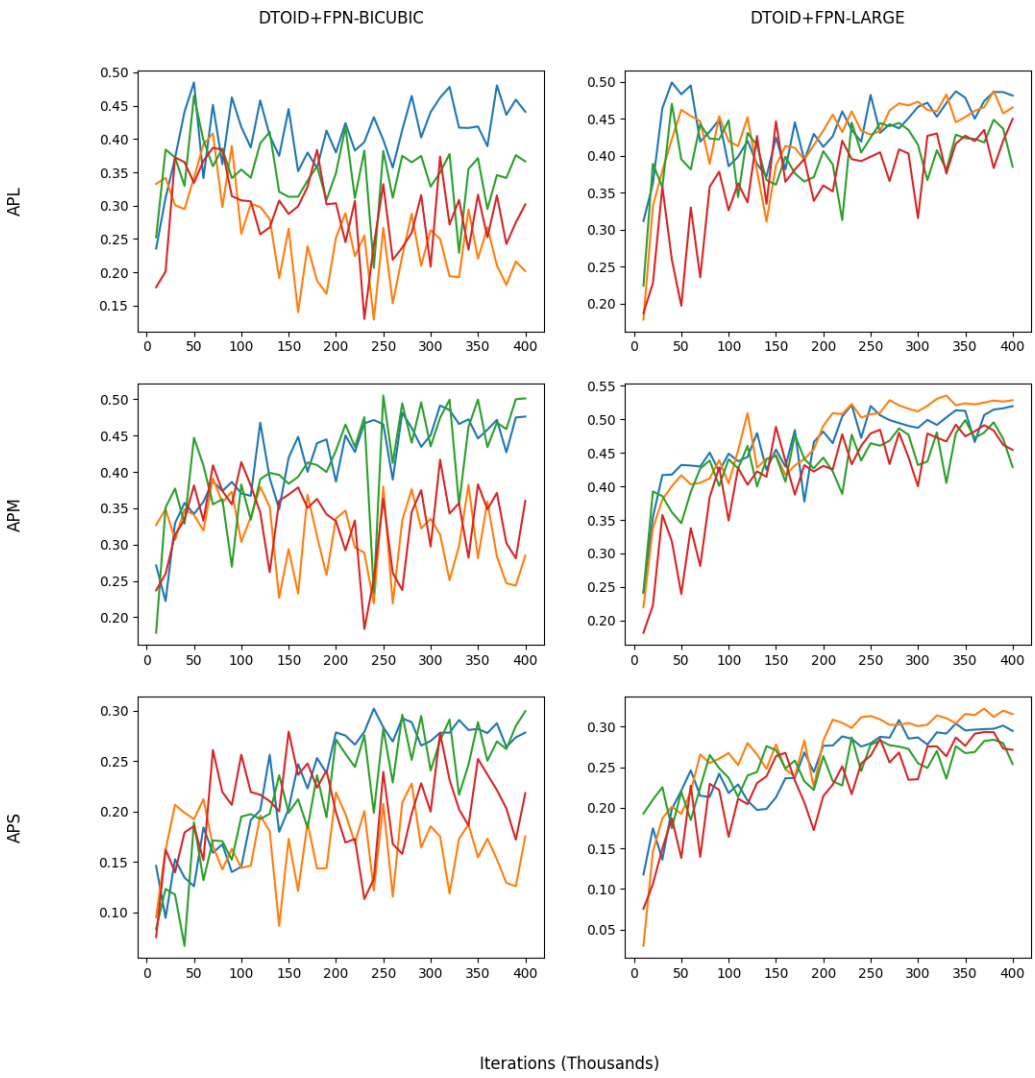

DTOID
DTOID-45R
DTOID-45B
APL
APM
APS
Iterations (Thousands)

B.2 DTOID+FPN

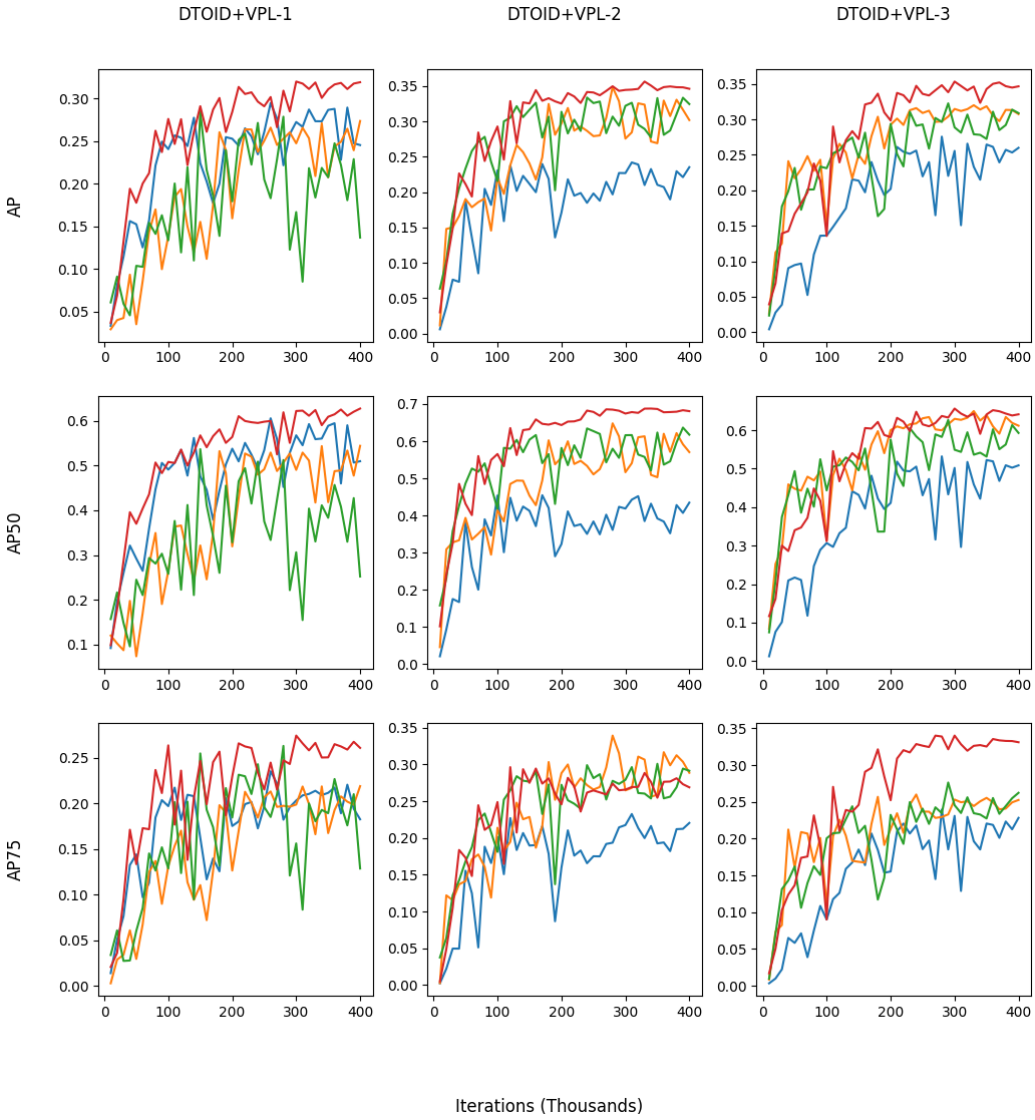

DTOID+FPN-B
DTOID+FPN-R
DTOID+FPN-CONCAT
AP
AP50
AP75
Iterations (Thousands)

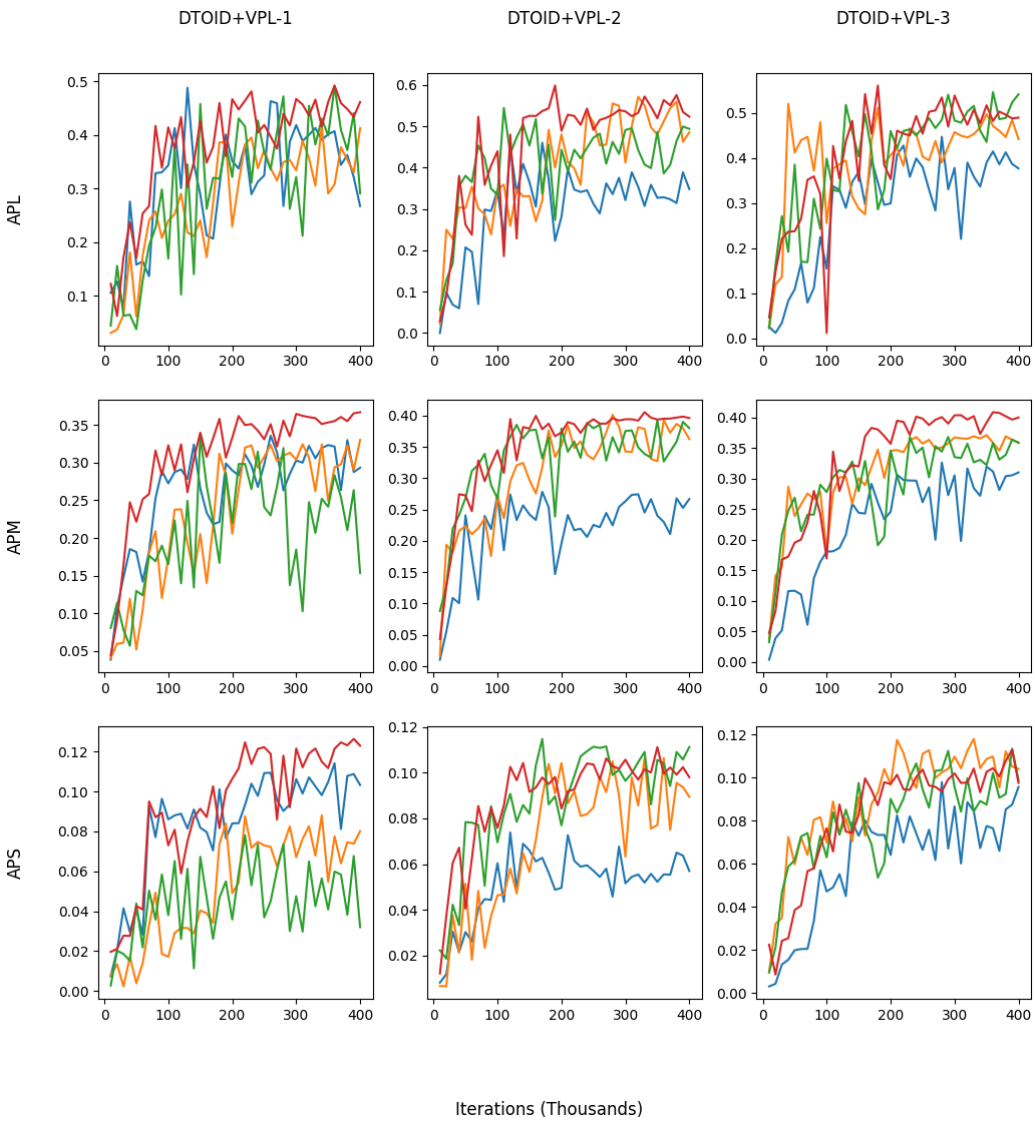

DTOID+FPN-B
DTOID+FPN-R
DTOID+FPN-CONCAT
APL
APM
APS
Iterations (Thousands)


DTOID+FPN-BICUBIC
DTOID+FPN-LARGE
AP
AP50
AP75
Iterations (Thousands)

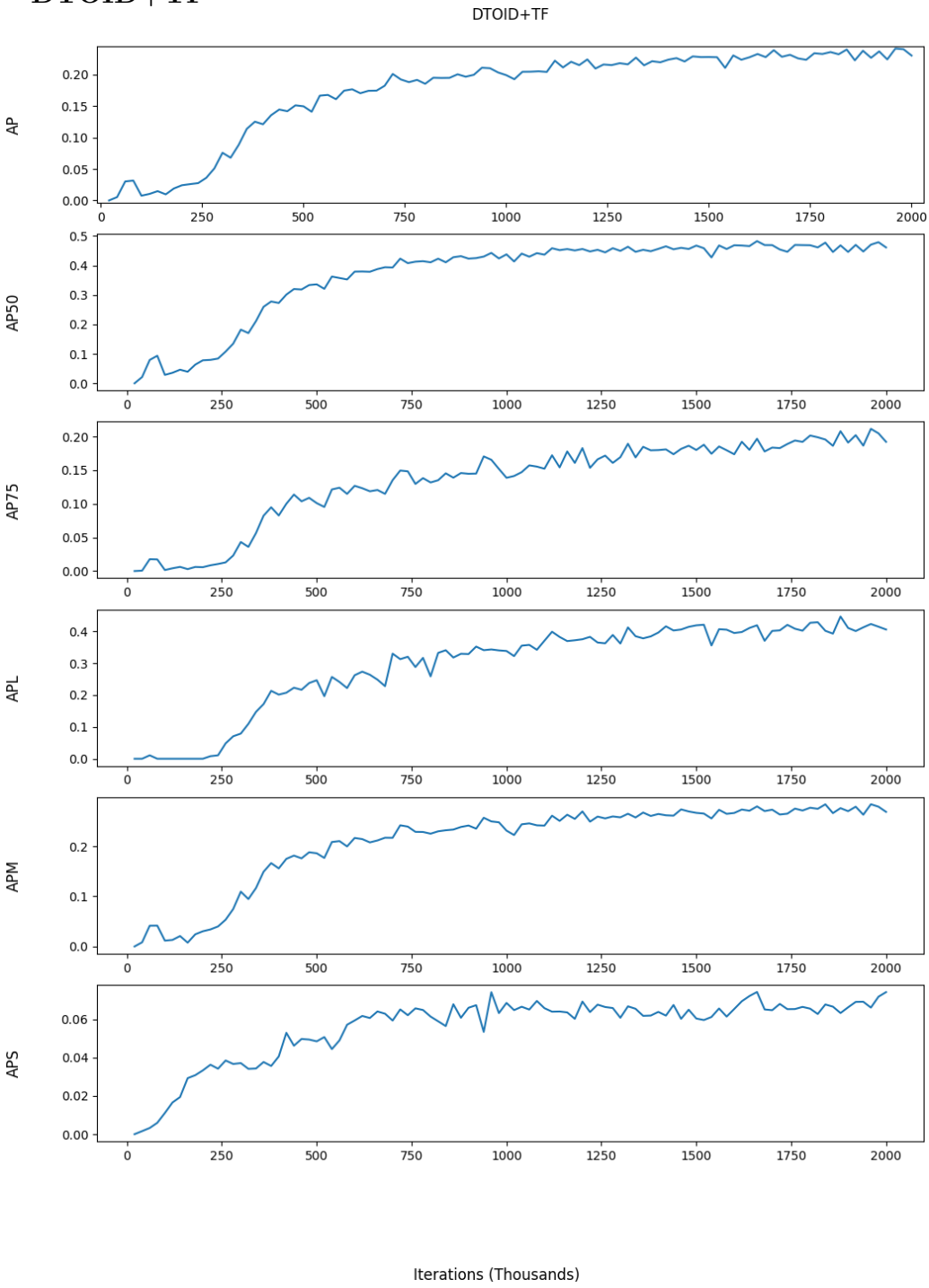

DTOID+FPN-BICUBIC
DTOID+FPN-LARGE
APL
APM
APS
Iterations (Thousands)

B.3 DTOID+VPL


DTOID+VPL-1
DTOID+VPL-2
DTOID+VPL-3
AP
AP50
AP75
Iterations (Thousands)


DTOID+VPL-1
DTOID+VPL-2
DTOID+VPL-3
APL
APM
APS
Iterations (Thousands)

B.4 DTOID+TF



APPENDIX C

BlenderProc Config

The provided config file was used with BlenderProc 1.10.0 for all image generation. It is intended to function as a reference for the generation procedure used rather than as a functional configuration. Several components are needed for functional usage, including the customized BOP Toolkit used, camera data, scripts to create segmentation maps from hd5f files and scene.blend file. The associated scene.blend file contains a single rectangular prism with a size of $0.4m \times 0.4m \times 2m$ placed at $0, 0, -1cm$ called Cube used for object placement. The other required files will be available in the used codebase should it be released in the future.

```
{
  "version": 3,
  "setup": {
    "blender_install_path": "/home_local/<env:USER>/blender/",
    "pip": [
      "h5py",
      "imageio", "scikit-image"
    ]
  },
  "modules": [
    {
      "module": "main.Initializer",
      "config": {
        "global": {
          "output_dir": "<args:0>",
          "output_is_temp": True,
          "sys_paths": ["../../../src/lib/bop_toolkit"]
        }
      }
    },
    {
      "module": "loader.BlendLoader",
      "config": {
        "path": "<args:1>",
        "load_from": "/Object",
        "entities": ".*"
      }
    },
    {
      "module": "manipulators.WorldManipulator",
      "config": {
        "cf_set_world_category_id": 0
      }
    },
    {
      "module": "lighting.LightSampler",
      "config": {
        "lights": [
        {
          "location": {
            "provider": "sampler.Shell",
            "center": [0, 0, 1],
            "radius_min": 0.5,
            "radius_max": 0.9,
            "elevation_min": 30,
            "elevation_max": 89.999,
            "uniform_elevation": True
          },
          "color": {
            "provider": "sampler.Color",
            "min": [0.8, 0.8, 0.8, 1.0],
            "max": [1.0, 1.0, 1.0, 1.0]
          },
          "type": "POINT",
          "energy": 10
        }
        ]
      }
    },
    {
      "module": "constructor.BasicMeshInitializer",
      "config": {
        "meshes_to_add": [
        {
          "type": "plane",
          "name": "ground_plane0",
          "scale": [1, 1, 1],
          "location": [0, 0, 1]
        },
        {
          "type": "plane",
          "name": "ground_plane1",
          "scale": [1, 0.5, 1],
          "location": [0, -1, 1.5],
          "rotation": [-1.570796, 0, 0]
        },
        {
          "type": "plane",
          "name": "ground_plane2",
          "scale": [1, 0.5, 1],
          "location": [0, 1, 1.5],
          "rotation": [1.570796, 0, 0]
        },
        {
          "type": "plane",
          "name": "ground_plane4",
          "scale": [0.5, 1, 1],
          "location": [1, 0, 1.5],
          "rotation": [0, -1.570796, 0]
        },
```

```
          {
            "type": "plane",
            "name": "ground_plane5",
            "scale": [0.5, 1, 1],
            "location": [-1, 0, 1.5],
            "rotation": [0, 1.570796, 0]
          },
          ]
        }
      },
      {
        "module": "manipulators.EntityManipulator",
        "config": {
          "selector": {
            "provider": "getter.Entity",
            "conditions": {
              "name": ".*"
            }
          },
          "cp_physics": False,
          "cp_category_id": 0
        }
      },
      {
        "module": "loader.BopLoader",
        "config": {
          "bop_dataset_path": "data/processed/bop_data/extemps_query",
          "model_type": "",
          "mm2m": True,
          "sample_objects": True,
          "num_of_objs_to_sample": <args:4>,
          "obj_instances_limit": 1,
          "split": "<args:3>",
          "obj_ids": <args:5>,
          "add_properties": {
            "cp_bop_dataset_name": "extemps_query",
            "cp_physics": True
          },
          "cf_set_shading": "SMOOTH"
        }
      },
      {
        "module": "manipulators.MaterialManipulator",
        "config": {
          "selector": {
            "provider": "getter.Material",
            "conditions": {
                  "name": "bop_extemps_query_vertex_col_material.*"
            }
          },
          "cf_set_specular": {
            "provider": "sampler.Value",
            "type": "float",
            "min": 0.0,
            "max": 1.0
          },
          "cf_set_roughness": {
            "provider": "sampler.Value",
            "type": "float",
            "min": 0.0,
            "max": 1.0
          }
        }
      },
      {
      "module": "object.OnSurfaceSampler",
      "config": {
        "objects_to_sample": {
          "provider": "getter.Entity",
          "conditions": {
            "cp_physics": True
          }
        },
        "surface": {
          "provider": "getter.Entity",
          "index": 0,
          "conditions": {
            "name": "Cube"
          }
        },
        "pos_sampler": {
          "provider": "sampler.UpperRegionSampler",
          "to_sample_on": {
            "provider": "getter.Entity",
            "index": 0,
            "conditions": {
              "name": "Cube"
            }
          },
          "min_height": 0.2,
```

```
          "max_height": 0.5,
          "use_ray_trace_check": False,
        },
        "min_distance": 0.02,
        "max_distance": 2,
        "rot_sampler": {
          "provider": "sampler.Uniform3d",
          "max": [0,0,0],
          "min": [6.28,6.28,6.28]
          }
        }
      },
      {
        "module": "object.PhysicsPositioning",
        "config": {
          "min_simulation_time": 3,
          "max_simulation_time": 10,
          "check_object_interval": 1,
          "solver_iters": 25,
          "friction": 100.0,
          "linear_damping": 0.99,
          "angular_damping": 0.99
        }
      },
      {
        "module": "camera.CameraSampler",
        "config": {
          "cam_poses": [
            {
              "number_of_samples": <args:2>,
              "location": {
                "provider": "sampler.Shell",
                "uniform_elevation": True,
                "center": [0, 0, 1],
                "radius_min": 0.6,
                "radius_max": 0.9,
                "elevation_min": 40,
                "elevation_max": 89.999
              },
              "rotation": {
                "format": "look_at",
                "value": [0, 0, 1],
                "inplane_rot": {
                  "provider": "sampler.Value",
                  "type": "float",
                  "min": -3.14159,
                  "max": 3.14159
                }
              }
            }
          ]
        }
      },
      # hide the sampling surface
      {
        "module": "object.ObjectPoseSampler",
        "config":{
          "max_iterations": 1,
          "objects_to_sample": {
            "provider": "getter.Entity",
            "index": 0,
            "conditions": {
              "name": "Cube"
            }
          },
          "pos_sampler": [100, 100, 0],
          "rot_sampler": [0, 0, 0]
        }
      },
      {
        "module": "renderer.SegMapRenderer",
        "config": {
          "map_by": ["instance", "class", "name"]
        }
      },
      {
        "module": "writer.Hdf5Writer",
        "config": {
          "output_is_temp": False,
          "append_to_existing_output": True,
          "output_dir": "data/interim/hdf5_query/<args:3>"
        }
      },
      {
        "module": "renderer.RgbRenderer",
        "config": {
          "samples": 50,
          "render_distance": True,
          "image_type": "JPEG"
        }
```

```
      },
      {
        "module": "writer.BopWriter",
        "config": {
          "dataset": "extemps_query",
          "append_to_existing_output": True,
          "output_is_temp": False,
          "depth_scale": 0.1,
          "ignore_dist_thres": 50.,
          "postprocessing_modules": {
            "distance": [
              {"module": "postprocessing.Dist2Depth"}
            ]
          }
        }
      }
    ]
  }
```

BIBLIOGRAPHY

"Blender," https://www.blender.org/download/releases/2-93/, accessed: 2022-05-07.

"Bop toolkit," https://github.com/thodan/bop_toolkit, accessed: 2022-05-07.

"Coco metrics," https://cocodataset.org, accessed: 2022-06-23.

"Osmesa," https://docs.mesa3d.org/osmesa.html, accessed: 2022-05-07.

Ammirato, P., Fu, C., Shvets, M., Kosecka, J., and Berg, A. C., "Target driven instance detection," *CoRR*, vol. abs/1803.04610, 2018. [Online]. Available: http://arxiv.org/abs/1803.04610

Bansal, A., Sikka, K., Sharma, G., Chellappa, R., and Divakaran, A., "Zero-shot object detection," *CoRR*, vol. abs/1804.04340, 2018. [Online]. Available: http://arxiv.org/abs/1804.04340

Bhiksha Raj, R. S., "Lecture notes on neural nets as universal approximators," 2021.

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

Chen, D.-J., Hsieh, H.-Y., and Liu, T.-L., "Adaptive image transformer for one-shot object detection," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2021, pp. 12 247–12 256.

Chen, H., Wang, Y., Wang, G., and Qiao, Y., "LSTD: A low-shot transfer detector for object detection," *CoRR*, vol. abs/1803.01529, 2018.

Chen, T., Liu, Y., Su, H., Chang, Y., Lin, Y., Yeh, J., and Hsu, W. H., "Should I look at the head or the tail? dual-awareness attention for few-shot object detection," *CoRR*, vol. abs/2102.12152, 2021.

Cybenko, G., "Approximation by superpositions of a sigmoidal function," *Mathematics of Control, Signals, and Systems (MCSS)*, vol. 2, no. 4, pp. 303–314, Dec. 1989. [Online]. Available: http://dx.doi.org/10.1007/BF02551274

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L., "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 2009, pp. 248–255.

Denninger, M., Sundermeyer, M., Winkelbauer, D., Zidan, Y., Olefir, D., Elbadrawy, M., Lodhi, A., and Katam, H., "Blenderproc," *CoRR*, vol. abs/1911.01911, 2019. [Online]. Available: http://arxiv.org/abs/1911.01911

Devlin, J., Chang, M., Lee, K., and Toutanova, K., "BERT: pre-training of deep bidirectional transformers for language understanding," *CoRR*, vol. abs/1810.04805, 2018. [Online]. Available: http://arxiv.org/abs/1810.04805

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J., and Houlsby, N., "An image is worth 16x16 words: Transformers for image recognition at scale," *CoRR*, vol. abs/2010.11929, 2020. [Online]. Available: https://arxiv.org/abs/2010.11929

Everingham, M., Gool, L., Williams, C. K., Winn, J., and Zisserman, A., "The pascal visual object classes (voc) challenge," *Int. J. Comput. Vision*, vol. 88, no. 2, p. 303–338, jun 2010. [Online]. Available: https://doi.org/10.1007/s11263-009-0275-4

Fan, Q., Zhuo, W., and Tai, Y., "Few-shot object detection with attention-rpn and multi-relation detector," *CoRR*, vol. abs/1908.01998, 2019.

Fan, Z., Ma, Y., Li, Z., and Sun, J., "Generalized few-shot object detection without forgetting," *CoRR*, vol. abs/2105.09491, 2021.

Girshick, R. B., "Fast R-CNN," *CoRR*, vol. abs/1504.08083, 2015. [Online]. Available: http://arxiv.org/abs/1504.08083

Girshick, R. B., Donahue, J., Darrell, T., and Malik, J., "Rich feature hierarchies for accurate object detection and semantic segmentation," *CoRR*, vol. abs/1311.2524, 2013. [Online]. Available: http://arxiv.org/abs/1311.2524

Han, G., Huang, S., Ma, J., He, Y., and Chang, S., "Meta faster R-CNN: towards accurate few-shot object detection with attentive feature alignment," *CoRR*, vol. abs/2104.07719, 2021.

He, K., Gkioxari, G., Dollár, P., and Girshick, R. B., "Mask R-CNN," *CoRR*, vol. abs/1703.06870, 2017. [Online]. Available: http://arxiv.org/abs/1703.06870

He, K., Zhang, X., Ren, S., and Sun, J., "Deep residual learning for image recognition," *CoRR*, vol. abs/1512.03385, 2015. [Online]. Available: http://arxiv.org/abs/1512.03385

He, K., Zhang, X., Ren, S., and Sun, J., "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," *CoRR*, vol. abs/1502.01852, 2015. [Online]. Available: http://arxiv.org/abs/1502.01852

Hinterstoisser, S., Holzer, S., Cagniart, C., Ilic, S., Konolige, K., Navab, N., and Lepetit, V., "Multimodal templates for real-time detection of texture-less objects in heavily cluttered scenes," in *2011 International Conference on Computer Vision*, 2011, pp. 858–865.

Hinterstoisser, S., Holzer, S., Cagniart, C., Ilic, S., Konolige, K., Navab, N., and Lepetit, V., "Multimodal templates for real-time detection of texture-less objects in heavily cluttered scenes," in *2011 International Conference on Computer Vision*, 2011, pp. 858–865.

Hodan, T., Michel, F., Brachmann, E., Kehl, W., Buch, A. G., Kraft, D., Drost, B., Vidal, J., Ihrke, S., Zabulis, X., Sahin, C., Manhardt, F., Tombari, F., Kim, T., Matas, J., and Rother, C., "BOP: benchmark for 6d object pose estimation," *CoRR*, vol. abs/1808.08319, 2018.

Hosang, J. H., Benenson, R., Dollár, P., and Schiele, B., "What makes for effective detection proposals?" *CoRR*, vol. abs/1502.05082, 2015.

Hu, H., Bai, S., Li, A., Cui, J., and Wang, L., "Dense relation distillation with context-aware aggregation for few-shot object detection," *CoRR*, vol. abs/2103.17115, 2021.

Huang, G., Liu, Z., and Weinberger, K. Q., "Densely connected convolutional networks," *CoRR*, vol. abs/1608.06993, 2016. [Online]. Available: http://arxiv.org/abs/1608.06993

Iandola, F. N., Moskewicz, M. W., Ashraf, K., Han, S., Dally, W. J., and Keutzer, K., "Squeezenet: Alexnet-level accuracy with 50x fewer parameters and <1mb model size," *CoRR*, vol. abs/1602.07360, 2016. [Online]. Available: http://arxiv.org/abs/1602.07360

Ioffe, S. and Szegedy, C., "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *CoRR*, vol. abs/1502.03167, 2015. [Online]. Available: http://arxiv.org/abs/1502.03167

Kamilaris, A. and Prenafeta-Boldú, F. X., "Deep learning in agriculture: A survey," *Computers and Electronics in Agriculture*, vol. 147, pp. 70–90, 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0168169917308803

Kang, B., Liu, Z., Wang, X., Yu, F., Feng, J., and Darrell, T., "Few-shot object detection via feature reweighting," *CoRR*, vol. abs/1812.01866, 2018. [Online]. Available: http://arxiv.org/abs/1812.01866

Kirkpatrick, J., Pascanu, R., Rabinowitz, N. C., Veness, J., Desjardins, G., Rusu, A. A., Milan, K., Quan, J., Ramalho, T., Grabska-Barwinska, A., Hassabis, D., Clopath, C., Kumaran, D., and Hadsell, R., "Overcoming catastrophic

forgetting in neural networks," *CoRR*, vol. abs/1612.00796, 2016. [Online]. Available: http://arxiv.org/abs/1612.00796

Köhler, M., Eisenbach, M., and Gross, H., "Few-shot object detection: A survey," *CoRR*, vol. abs/2112.11699, 2021.

Krizhevsky, A., Sutskever, I., and Hinton, G. E., "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems*, Pereira, F., Burges, C. J. C., Bottou, L., and Weinberger, K. Q., Eds., vol. 25. Curran Associates, Inc., 2012. [Online]. Available: https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf

LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., and Jackel, L. D., "Backpropagation applied to handwritten zip code recognition," *Neural Computation*, vol. 1, no. 4, pp. 541–551, 1989.

Li, X., Zhang, L., Chen, Y. P., Tai, Y., and Tang, C., "One-shot object detection without fine-tuning," *CoRR*, vol. abs/2005.03819, 2020.

Li, Z., Yang, W., Peng, S., and Liu, F., "A survey of convolutional neural networks: Analysis, applications, and prospects," *CoRR*, vol. abs/2004.02806, 2020.

Lin, T., Wang, Y., Liu, X., and Qiu, X., "A survey of transformers," *CoRR*, vol. abs/2106.04554, 2021. [Online]. Available: https://arxiv.org/abs/2106.04554

Lin, T., Dollár, P., Girshick, R. B., He, K., Hariharan, B., and Belongie, S. J., "Feature pyramid networks for object detection," *CoRR*, vol. abs/1612.03144, 2016. [Online]. Available: http://arxiv.org/abs/1612.03144

Lin, T., Goyal, P., Girshick, R. B., He, K., and Dollár, P., "Focal loss for dense object detection," *CoRR*, vol. abs/1708.02002, 2017. [Online]. Available: http://arxiv.org/abs/1708.02002

Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S. E., Fu, C., and Berg, A. C., "SSD: single shot multibox detector," *CoRR*, vol. abs/1512.02325, 2015.

Marcel, S. and Rodriguez, Y., "Torchvision the machine-vision package of torch," in *Proceedings of the 18th ACM International Conference on Multimedia*, ser. MM '10. New York, NY, USA: Association for Computing Machinery, 2010, p. 1485–1488. [Online]. Available: https://doi.org/10.1145/1873951.1874254

McCloskey, M. and Cohen, N. J., "Catastrophic interference in connectionist networks: The sequential learning problem," ser. Psychology of Learning and Motivation, Bower, G. H., Ed. Academic Press, 1989, vol. 24, pp. 109–165. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0079742108605368

Mercier, J.-P., Garon, M., Giguère, P., and Lalonde, J.-F., "Deep template-based object instance detection," 2019. [Online]. Available: https://arxiv.org/abs/1911.11822

Mhaskar, H. N., Liao, Q., and Poggio, T. A., "Learning real and boolean functions: When is deep better than shallow," *CoRR*, vol. abs/1603.00988, 2016. [Online]. Available: http://arxiv.org/abs/1603.00988

Michaelis, C., Ustyuzhaninov, I., Bethge, M., and Ecker, A. S., "One-shot instance segmentation," *CoRR*, vol. abs/1811.11507, 2018.

Mitchell, T., *Machine Learning*. McGraw-Hill Education, 1997.

Murphy, K. P., *Probabilistic Machine Learning: An introduction*. MIT Press, 2022. [Online]. Available: probml.ai

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S., "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*, Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., and Garnett, R., Eds. Curran Associates, Inc., 2019, pp. 8024–8035. [Online]. Available: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

Rahman, S., Khan, S. H., and Porikli, F., "Zero-shot object detection: Learning to simultaneously recognize and localize novel concepts," *CoRR*, vol. abs/1803.06049, 2018. [Online]. Available: http://arxiv.org/abs/1803.06049

Redmon, J. and Farhadi, A., "YOLO9000: better, faster, stronger," *CoRR*, vol. abs/1612.08242, 2016.

Ren, S., He, K., Girshick, R. B., and Sun, J., "Faster R-CNN: towards real-time object detection with region proposal networks," *CoRR*, vol. abs/1506.01497, 2015. [Online]. Available: http://arxiv.org/abs/1506.01497

Rosenblatt, F., "The perceptron - a perceiving and recognizing automaton," Cornell Aeronautical Laboratory, Ithaca, New York, Tech. Rep. 85-460-1, January 1957.

Schwartz, E., Karlinsky, L., Shtok, J., Harary, S., Marder, M., Pankanti, S., Feris, R. S., Kumar, A., Giryes, R., and Bronstein, A. M., "Repmet: Representative-based metric learning for classification and one-shot object detection," *CoRR*, vol. abs/1806.04728, 2018. [Online]. Available: http://arxiv.org/abs/1806.04728

Shmelkov, K., Schmid, C., and Alahari, K., "Incremental learning of object detectors without catastrophic forgetting," *CoRR*, vol. abs/1708.06977, 2017. [Online]. Available: http://arxiv.org/abs/1708.06977

Simonyan, K. and Zisserman, A., "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

Sun, B., Li, B., Cai, S., Yuan, Y., and Zhang, C., "FSCE: few-shot object detection via contrastive proposal encoding," *CoRR*, vol. abs/2103.05950, 2021. [Online]. Available: https://arxiv.org/abs/2103.05950

Sun, C., Shrivastava, A., Singh, S., and Gupta, A., "Revisiting unreasonable effectiveness of data in deep learning era," *CoRR*, vol. abs/1707.02968, 2017.

Tjaden, H., Schwanecke, U., and Schömer, E., "Real-time monocular pose estimation of 3d objects using temporally consistent local color histograms," in *2017 IEEE International Conference on Computer Vision (ICCV)*, 2017, pp. 124–132.

Uijlings, J. R., Sande, K. E., Gevers, T., and Smeulders, A. W., "Selective search for object recognition," *Int. J. Comput. Vision*, vol. 104, no. 2, p. 154–171, sep 2013. [Online]. Available: https://doi.org/10.1007/s11263-013-0620-5

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I., "Attention is all you need," *CoRR*, vol. abs/1706.03762, 2017.

Vinyals, O., Blundell, C., Lillicrap, T. P., Kavukcuoglu, K., and Wierstra, D., "Matching networks for one shot learning," *CoRR*, vol. abs/1606.04080, 2016.

Wang, Q., Zhang, L., Bertinetto, L., Hu, W., and Torr, P. H., "Fast online object tracking and segmentation: A unifying approach," *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun 2019. [Online]. Available: http://dx.doi.org/10.1109/CVPR.2019.00142

Wang, X., Huang, T. E., Darrell, T., Gonzalez, J. E., and Yu, F., "Frustratingly simple few-shot object detection," *CoRR*, vol. abs/2003.06957, 2020. [Online]. Available: https://arxiv.org/abs/2003.06957

Wu, J., Liu, S., Huang, D., and Wang, Y., "Multi-scale positive sample refinement for few-shot object detection," *CoRR*, vol. abs/2007.09384, 2020. [Online]. Available: https://arxiv.org/abs/2007.09384

Wu, X., Sahoo, D., and Hoi, S. C. H., "Recent advances in deep learning for object detection," *Neurocomputing*, vol. 396, pp. 39–64, 2020.

Xian, Y., Lampert, C. H., Schiele, B., and Akata, Z., "Zero-shot learning - A comprehensive evaluation of the good, the bad and the ugly," *CoRR*, vol. abs/1707.00600, 2017. [Online]. Available: http://arxiv.org/abs/1707.00600

Xiao, J., Owens, A., and Torralba, A., "Sun3d: A database of big spaces reconstructed using sfm and object labels," in *2013 IEEE International Conference on Computer Vision*, 2013, pp. 1625–1632.

Xiao, Y. and Marlet, R., "Few-shot object detection and viewpoint estimation for objects in the wild," *CoRR*, vol. abs/2007.12107, 2020.

Yan, X., Chen, Z., Xu, A., Wang, X., Liang, X., and Lin, L., "Meta R-CNN : Towards general solver for instance-level low-shot learning," *CoRR*, vol. abs/1909.13032, 2019. [Online]. Available: http://arxiv.org/abs/1909.13032

Yang, J., Li, S., Wang, Z., Dong, H., Wang, J., and Tang, S., "Using deep learning to detect defects in manufacturing: A comprehensive survey and current challenges," *Materials*, vol. 13, no. 24, 2020. [Online]. Available: https://www.mdpi.com/1996-1944/13/24/5755

Yang, Y., Wei, F., Shi, M., and Li, G., "Restoring negative information in few-shot object detection," *CoRR*, vol. abs/2010.11714, 2020. [Online]. Available: https://arxiv.org/abs/2010.11714

Yang, Z., Li, J., and Li, H., "Real-time pedestrian detection for autonomous driving," in *2018 International Conference on Intelligent Autonomous Systems (ICoIAS)*, 2018, pp. 9–13.

Zacarias, A. S. and Alexandre, L. A., "Overcoming catastrophic forgetting in convolutional neural networks by selective network augmentation," *CoRR*, vol. abs/1802.08250, 2018. [Online]. Available: http://arxiv.org/abs/1802.08250

Zeiler, M. D. and Fergus, R., "Visualizing and understanding convolutional networks," *CoRR*, vol. abs/1311.2901, 2013. [Online]. Available: http://arxiv.org/abs/1311.2901

Zhang, G., Cui, K., Wu, R., Lu, S., and Tian, Y., "Pnpdet: Efficient few-shot detection without forgetting via plug-and-play sub-networks," in *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, January 2021, pp. 3823–3832.

Zhang, G., Luo, Z., Cui, K., and Lu, S., "Meta-detr: Few-shot object detection via unified image-level meta-learning," *CoRR*, vol. abs/2103.11731, 2021. [Online]. Available: https://arxiv.org/abs/2103.11731

Zhou, Q. and Jacobson, A., "Thingi10k: A dataset of 10,000 3d-printing models," *arXiv preprint arXiv:1605.04797*, 2016.