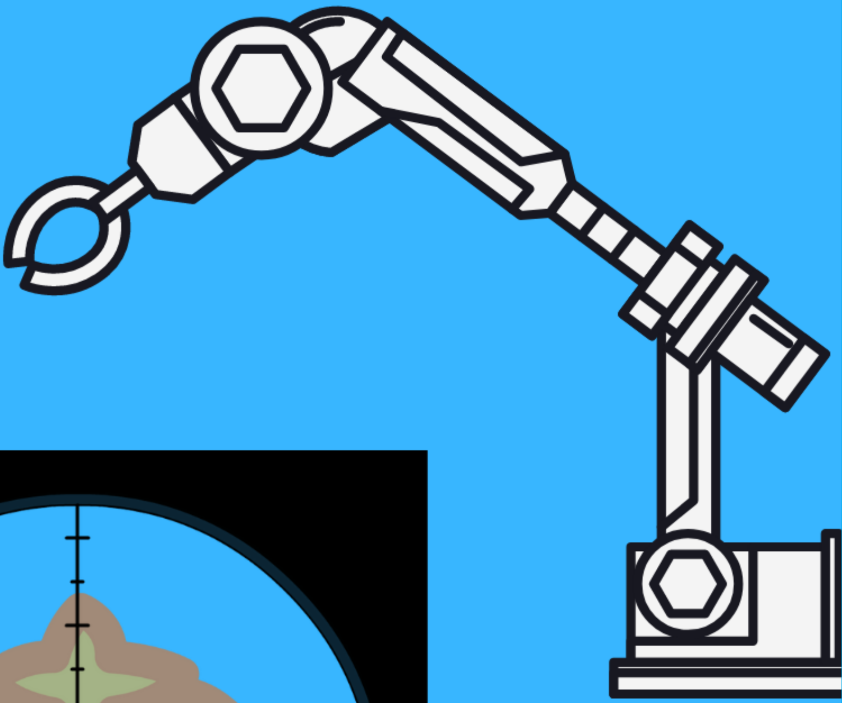
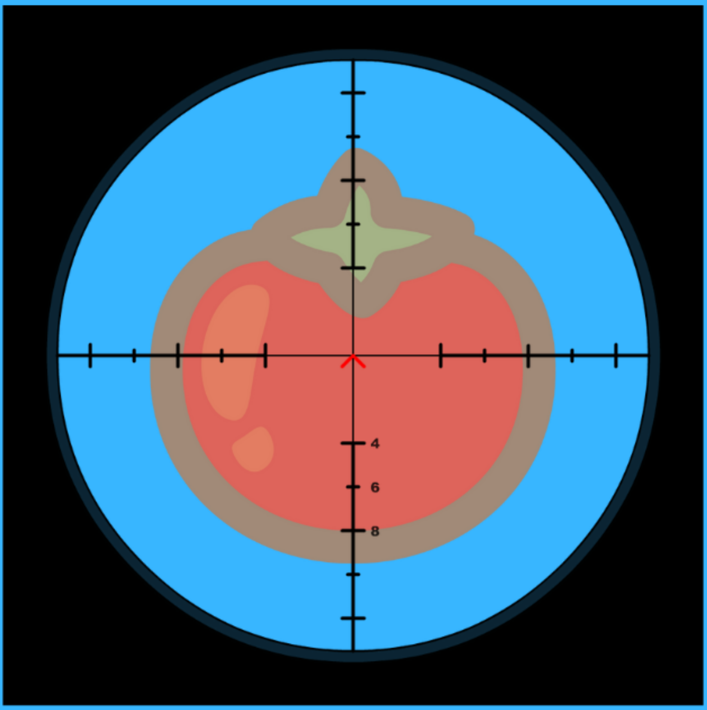
Delft University of Technology

DATA EFFICIENT PICK LOCATION LEARNING FROM IMAGES FOR SEQUENTIAL TASKS

KARTHIK ARVIND ARUNMOLI





TU Delft

Data efficient pick location learning from images for sequential tasks

by

Karthik Arvind Arunmoli

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Friday October 29, 2021 at 14:00.

Student number:	5062918	
Project duration:	December 1, 2020 – October 29, 2021	
Thesis committee:	Prof. dr. ir. J. Kober,	TU Delft, supervisor
	Dr. J.F.P Kooij,	TU Delft, External Committee
	Ir. R. Perez,	TU Delft, supervisor

This thesis is confidential and cannot be made public until October 29, 2021.

Preface

The two years of my master's program in Delft has been an eye-opening experience for me, both academically and personally. Going through my thesis during the pandemic was even more challenging. The life experience that I got through my master's is irreplaceable and it would not be possible without the support of many individuals.

I would firstly like to thank Prof.J.Kober for giving me an opportunity for doing a master's thesis under him and FLEXCRAFT research program. His insights were always eye-opening. Secondly I would like to thank my daily supervisor Ir.Rodrigo Perez. His guidance and support through the thesis has had a huge impact on me, both professionally and mentally during the course of my thesis.

I would like to thank my family for their exceptional support. My Mom, Dad and my sister for their motivation and constant push. I would like to thank my girl friend and my friends for their constant support and advises.

Lastly, I would like to dedicate this thesis to my late grandmother.

Karthik Arvind Arunmoli
Delft, October 2021

List of Figures

List of Tables

Abstract

Learning from demonstration is a technique where the robot learns directly from humans. It can be beneficial to learn from humans directly because humans can easily demonstrate complex behaviors without being experts in demonstrating required tasks. However, it can be challenging to gather large amounts of data from humans because humans often get tired, bored, or lose focus while giving many demonstrations. Therefore, the learning algorithms require to be as data-efficient as possible. To solve sequential tasks using few human demonstrations, the Transporter Network was introduced.

The Transporter Network consists of two networks, namely the pick network and the place network. Both these networks have a fully convolutional architecture which has the property of being translationally equivariant. Translational equivariance is well suited for estimating pick locations because the predicted pick location changes corresponding to the change in the object's position in the image. However, the performance of the transporter network on sequential tasks depends on the receptive field of the fully convolutional network because each pixel in the output should depend on each pixel in the input image. This correlation is important for the network to make predictions based on the overall configuration of objects in the input image. A larger receptive field would give a prediction by correlating large amount of pixels in the input image, however increasing the receptive field could result in loss of resolution which in turn leads to information loss in the latent space. Therefore, it is essential to select an appropriate size for the receptive field such that there is minimal loss of resolution in the latent space of the fully convolutional network. This limitation of selecting the appropriate size of the receptive field was also observed in the preliminary experiments on the pick network of the Transporter Network.

In this work, a SEquential Attention Network (SEA Net) is introduced to overcome the dependence of the Transporter Network on the receptive field size for solving sequential task. SEA Net is a variation of the pick network of the Transporter Network. SEA Net works under the assumption that the set of objects to be picked are known beforehand. Based on this assumption, the SEA Net learns to predict two things: 1) The pick location of all the objects. 2) "what" object to pick as a function of the current configuration of the objects in the environment. SEA Nets can be used independently to predict pick location sequentially and also be used as a replacement of the pick network in the Transporter Network. SEA Net is evaluated on two datasets, namely synthetic dataset and simulated robot dataset. The synthetic dataset has a top-down view of simple shape blocks. In contrast, simulated robot datasets are extracted using a pre-trained policy on the Mujoco simulators. The Object Keypoint Similarity (OKS) metrics is used to score the distance between the predicted pick point and the ground truth. The OKS Metric uses a standard deviation threshold ($\sigma_{threshold}$). A strict standard deviation threshold ($\sigma_{threshold}$) of 2 pixels and a lenient threshold ($\sigma_{threshold}$) of 20 pixels is used because a strict threshold will evaluate the resolution of the prediction. In contrast, the lenient threshold will assess if the predicted pick point is, at least on the desired object. The model achieved an overall accuracy of 64% for a strict standard deviation threshold ($\sigma_{threshold}$) of 2 pixels and accuracy of 85% for a lenient standard deviation threshold ($\sigma_{threshold}$) of 20 pixels on the OKS metrics, on the synthetic dataset. The model could perform on a simulated robot dataset with an accuracy of 82% on a strict standard deviation threshold ($\sigma_{threshold}$) of 2 pixels.

Contents

1 Introduction

The need to introduce robots for automated process has paved the way for new technologies. There has been an exponential increase in the use of robots in various areas [1, 2]. For example, logistical warehouses use robots for sorting packages. The field of agriculture also uses robots for harvesting fruits and vegetables. Incorporating robots in industries has paved the way to higher quality in production and reduced costs [3]. Robots can react autonomously to the changes in the environment due to unforeseen situations. To achieve autonomy, the robots need to be trained to perform a specific task. The training of the robot could be based on cameras and other sensors. Cameras provide images and teaching a robot using images is a complex and systematic problem because they are high-dimensional input and require high memory and computation techniques to get the desired results. However, cameras(RGB) are cheap and provide rich information about the surrounding of the robot. The advancement in deep learning has increased the interest of using images to extract relevant information from images. Hence the thesis aims to train a robot to perform data-efficient, sequential tasks using images. The thesis will focus on estimating the pick location of a sequential task as a function of an image that provides relevant information about the environment in each time step.

1.1. Motivation

Learning to execute sequential tasks from demonstration is an interesting approach to train the robot easily. In learning from demonstrations, the robot can learn from human demonstrations directly. It is beneficial to use human demonstrations because humans can demonstrate complex behaviors without being an expert in robotics [4]. However, it is not possible to expect humans to provide a large amount of demonstration because humans can often get bored, tired, or lose focus while giving the demonstration. Therefore, it is essential to learn the sequential tasks with fewer demonstrations. To learn a sequential task data-efficiently, a hierarchy could be introduced into the problem. By introducing a hierarchy, a structure is introduced into the sequential task, which will be better for generalizing in unseen scenarios. Therefore, sequential tasks could be learned using hierarchical learning algorithms. In the context of imitation learning, a hierarchical learning algorithm breaks the overall demonstrations into sub-tasks, thereby reducing the size of the state space for each sub-task [5]. Hierarchical learning aims to learn high-level and low-level behavior together (as explained in Appendix A.1). In the context of robotics, for instance, a high-level controller can provide goals to be reached by a manipulator, while the low-level controller sends to the robot the control commands to follow a trajectory that will take the robot from its current position to these requested goals. The high-level goals can be a desired robot's state, like a joint-state configuration or an end-effector position. A low-level controller can use motion primitives to move the robot from one state to another. Therefore, a combination of the high-level and low-level controller could execute a sequential task. This thesis will focus on learning the high-level goals of the mentioned hierarchical structure, assuming that the low-level controller already exists.

To extract high-level goals from images deep learning approaches could be used. It is possible to extract high-level goals as poses or spatial locations from images using computer-vision techniques like object detection, image segmentation, etc [6, 7]. It is also possible to use convolutional neural networks to directly map the image to the required goal poses or spatial locations. Some of these deep learning techniques are already designed to extract high-level goal locations from image for robotic manipulation tasks [8–10]. However, the limitation of using these techniques for extracting high-level goal, is its inability to estimate the exact spatial location for the robot to reach using limited data. The model needs to perform well using fewer demonstration as they are provided by humans.

Hence the aim is to extract the spatial locations of desired high-level goals from images to carry out a sequential task with few demonstrations. To be more clear on the objective, consider a robot that needs to move from location 1 to 4 through intermediate points 2, and 3 Fig 1.1. It could be assumed that the robot carries out a sequential task, i.e., it reaches point 1 first, then 2, etc. The aim is to predict the robot's high-level goals that are the spatial location of the points 1, 2, 3, and 4 which needs to be predicted consecutively with fewer demonstrations. The main motivation is to devise a computer-vision technique that could learn to predict the spatial location of the goals for a sequential tasks with limited data.

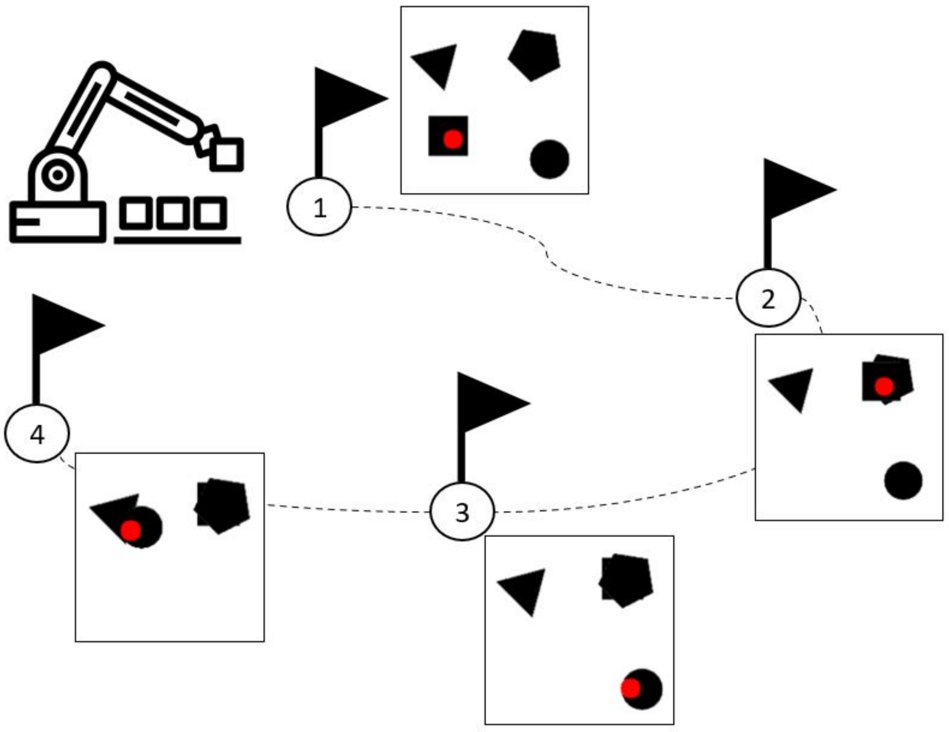


Figure 1.1: The objective is to extract the target positions the robot needs to reach from images.

1.2. Aim and Objectives

To solve the sequential task considered in this thesis, the high-level controller needs to provide high-level goals in the form of spatial locations. Each spatial location corresponds to a pick point in a manipulation task. Consider an example of a tomato harvester. The task of the tomato harvester is to pick up a cutting tool, harvest a tomato, and place the tomato in a box. Here the goal of the high-level controller is to give the goal location/pick location during each of the four tasks from images. The challenge lies in learning what high-level goal is needed for the low-level controller, along with the goal location, to manipulate the sequential task with fewer demonstrations successfully. Some deep learning techniques use a fully convolutional network to solve sequential tasks with fewer demonstrations. These fully convolutional networks have an hourglass structure in which the output is predicted in the same dimension of the input by correlating with each pixel in the input. Hence, it is essential to have large receptive fields to make predictions incorporating more information from the image. However, increasing the receptive field means increasing the loss of information in the latent space and the parameter, which affects the generalization of the model with few datasets. Hence, a trade-off is always set between maximizing the receptive field and minimizing the loss of information in predicting the output.

Hence it is required to develop a model that is independent of this limitation so that it is possible to consecutively predict the goal/pick locations without any loss of information in the latent space. This goal was divided into a set of objectives listed below

- Define and implement a method that can predict high-level goals in the form of pick locations in sequential tasks data-efficiently and removing the trade-off present in Fully Convolutional Hourglass Networks between their receptive field size and the loss of information in the latent space.
- Create and extract simulated robot dataset and synthetic dataset.
- Validate the method on simulated robot dataset and synthetic dataset.

1.3. Outline

The content of this thesis will have the following outline:

- Chapter 2 introduces concepts and literature to understand the thesis in detail. It will start from the basic concept of neural networks and move towards the literature that are necessary to understand the thesis implementation.
- Chapter 3 explains the novel concept of multi-output pick Network which is introduced to overcome the problem faced in current literature to solve sequential pick and place task. The architecture and the loss function used are introduced in this chapter.
- Chapter 4 explains the preliminary experiments carried out on the pick network of Transporter Network and the multi-output pick network.
- Chapter 5 will propose the architecture to solve sequential prediction of pick locations. The chapter will introduce the architecture and the loss function of the proposed model.
- Chapter 6 will introduce the dataset and experiments used to evaluate the performance of the proposed methods. It also gives an analysis of the results obtained.

2

Theoretical Background and Related works

In this chapter, an overview of the fundamental concepts are provided for understanding the core of this project. The basic concept in deep learning are explained followed by the fundamentals of Convolutional Neural Networks (CNNs). After explaining the fundamentals of CNNs, the hourglass structure is introduced. Then the various method used in the current literature for extracting pick points and the metrics used to evaluate the performance of the model are discussed.

2.1. Deep Learning Background

Artificial intelligence is used widely to make machines behave like humans. One of the widely used subsets of Artificial intelligence is Machine learning, where the decision is made based on the given features of the dataset. Deep learning is a subset of Machine learning in which the model has the inherent capability to learn the feature along with the decisions. This property of deep learning has become a powerful tool in many applications and is exploited in various decision-making frameworks. The deep learning model is trained to learn a function that maps an input(x) to an output(y) based on the Eq (2.1)

$$y = f(x, \theta) \tag{2.1}$$

where,

θ is the model parameters

2.1.1. Neural Networks

Neural networks or feed-forward neural networks correlates the input to the output by approximating some function f^*. These networks are called feed-forward neural networks because the information flows from the input to the output without any feedback. It is possible to use different functions to map the input to the output as represented in Eq (2.2).

$$f(x) = f^{(3)}\left(f^{(2)}\left(f^{(1)}(x)\right)\right) \tag{2.2}$$

where:

$$f^{(1)} \text{ is the first layer of the network}$$
$$f^{(2)} \text{ is the hidden layer of the network}$$
$$f^{(3)} \text{ is the output layer of the network}$$

Neural networks are called neural because they are inspired by neurons in neural science. Therefore, the building block of neural networks is the neurons. During training, each neuron's weight and bias value are stored as shown in Eq (2.3), and the final output is calculated as the weighted sum of the weight and biases [11]. The neurons use a linear model, and the non-linearity of the input could be represented by using activation functions.

$$f(x; w, b) = \sum x^{\top} w + b \tag{2.3}$$

where:

$$w = \text{Weight parameter}$$
$$b = \text{Bias parameter}$$
$$x = \text{input to the network}$$

From the above equation, the value of the function f could range from -inf to +inf. There is no mean for the neuron to know the bound which lead to its firing. An *activation function* was introduced to help the neurons understand the bounds for firing. The activation function is used to convert the linear output of the neuron to non-linear output that is easy for computation. The activation functions are also differentiable making it easy for updating the parameters during backward propagation [12]. The activation also helps control the output value passed to the calculation, thereby reducing the computational burden. There are linear and non-linear activation functions of which only the main non-linear functions are discussed. The different type of activation functions are depicted in Fig 2.1.

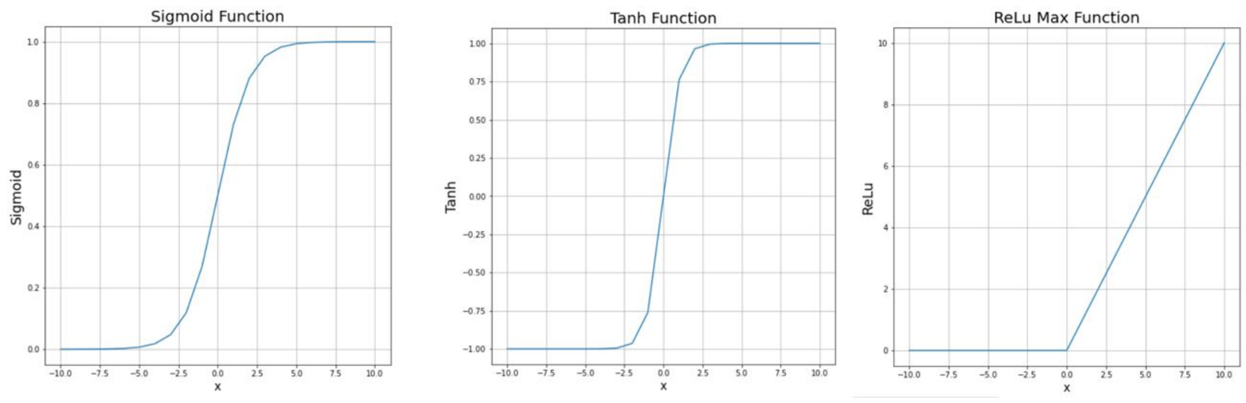


Figure 2.1: The different types of activation functions

Sigmoid functions are referred to as squashing functions used to keep the value bounds between 0 and 1 [13]. The sigmoid function is represented as in the Eq (2.4).

$$f(x) = \left(\frac{1}{(1 + \exp^{-x})} \right) \tag{2.4}$$

Since the value is bounded between 0 and 1, the sigmoid functions could predict outputs based on probability. Hence the wide application of sigmoid function is in binary classification problems. The major drawbacks of using a sigmoid functions are vanishing gradient problem during backpropogation and a non-zero centered function [12]. This problem is tackled in other activation functions.

Hyperbolic Tangent Function (*tanh*) is like a sigmoid function, but will squash the outputs between the value of -1 and 1. It is a zero-centered function, thereby adding in the computation during backward propagation. The tanh function is represented in Eq (2.5)

$$f(x) = \left(\frac{e^x - e^{-x}}{e^x + e^{-x}}\right) \tag{2.5}$$

Softmax function is another activation function that is used to scale the output between 0 and 1. The main property of a softmax function is that sum of the output probabilities is equal to 1. It is represented in the Eq (2.6).

$$f\left(x_i\right) = \frac{\exp\left(x_i\right)}{\sum_j \exp\left(x_j\right)} \tag{2.6}$$

The softmax activation function is mainly used in multi-class models in which the probability of each class is given with the highest probability corresponding to the target class.

Rectified Linear Unit (*ReLu*) is the most used activation function that is used in various deep learning models. The Relu activation was introduced to eliminate the vanishing gradient problems of sigmoid and tanh activation functions [14]. The ReLu activation eliminates the negative value and keeps only the positive value as shown in Eq (2.7).

$$\mathrm{relu}(x) = \max(0, x) \tag{2.7}$$

ReLu is advantageous in computing the gradients as the gradient for smaller values is 0, and the gradients of larger values are 1. However, eliminating the negative values means there is no contribution by these values to improving the network. This problem is called the *dying ReLu* problem and was the motivation for the introduction of Leaky ReLu and parameterized ReLu [14].

After the values are passed through the output layer, the neural network evaluates the loss function. The loss function computes the error between the predicted output and the ground truth. This scalar$J(\theta)$ error is used to update the model parameters such that eventually, a global minimum error is obtained. Many loss functions could be used, and the choice of the loss function depends upon the task at hand. The generalized depiction of the loss function is shown in Eq (2.8).

$$J(\theta) = L_f(\hat{y}, y) \tag{2.8}$$

where,

$\hat{y}$ = prediction of the network
y = ground truth
L_f = Loss function

The model parameters are usually updated based on *back-propagation algorithm*. This is a method that is used for computing the gradient of the model parameters for a particular scalar error. The optimiser algorithms like the stochastic gradient descent (SGD) is used to learn using the gradient of a back-propagation algorithms [11].

2.1.2. Convolutional Neural Network (*CNNs*)

Convolutional neural networks are the most popular networks used in image processing because of their ability to reduce the number of training parameters in the learning network [15]. One of the most important features of using a CNN network is its spatial independence i.e.; the CNN detects a feature regardless of its location in the image.

The CNNs get their name because of the use of a mathematical linear operator called convolution. Convolution operation in mathematics is similar to cross-correlation operation and is depicted by Eq (2.9).

$$G[i,j] = \sum_{u=-k}^{k} \sum_{v=-k}^{k} H[u,v]F[i-u,j-v] \tag{2.9}$$

where

$G[i,j]$ is the output in the next layer
$F[i-u,j-v]$ is the filter or kernel matrix
$H[u,v]$ is the input image

Why CNNs ?: Using a fully connected layer for processing an image will require many parameters to be learned because each pixel in the image needs to be connected to the next layer. However, the introduction of convolution drastically decreases the number of parameters required because of local patch attention, i.e., a local patch of the image is connected to the neuron in the next layer. In addition, in a convolution network, the weight is shared/fixed for the neurons for the entire layer. This property decreases the parameters, further making the model more efficient.

The visualization of the modern CNN network is shown in Fig 2.2.

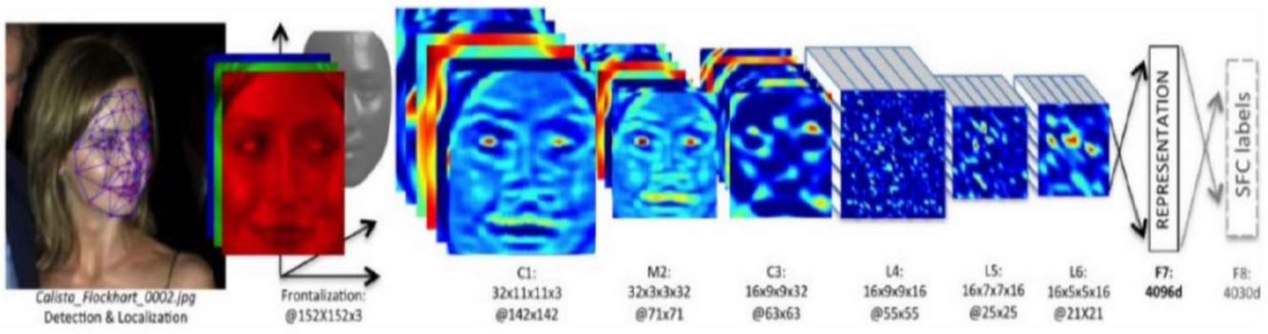


Figure 2.2: The visualisation of the CNN filters for an image recognition application [15]

Various operations are integrated in the CNN for reducing the parameters more and more. These are discussed in the following section.

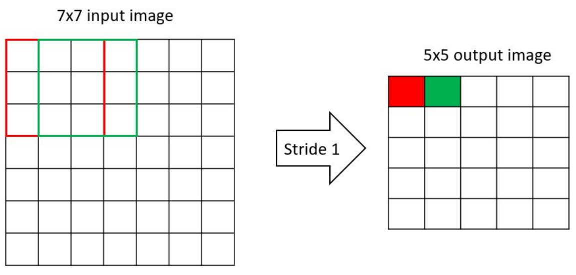
Stride and Padding

Stride operation is used to control the overlap of the regions in the image with the neighboring neuron. An example of the stride operation is depicted in Fig 2.3. It is seen that with an increase in stride value, the size of the output is also decreased.

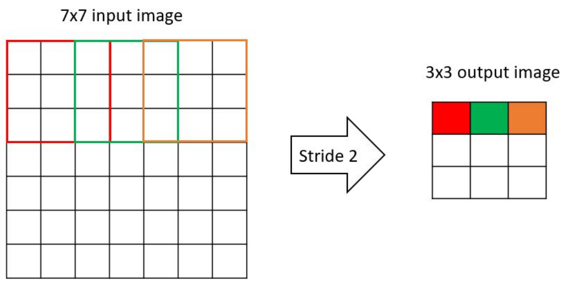
While extracting information to the next layer connection through stride, it is not possible to extract the data in the borders of the image as the filter will not pass through this region of the image. This disadvantage of the CNNs are overlooked using a padding operation. **Padding** operation is used to compensate the loss of the image data in the corners of the image. The simple way to do this is by adding zero-padding, i.e., adding zeros in the sides of the image. Padding is also used to control the output size.

Pooling

The main idea of pooling is to reduce the resolution of the input image based on the values present in the neighborhood so that the next computation has fewer parameters to be computed. For example, as depicted in Fig 2.4, the maximum output within the rectangular neighborhood is taken for the



(a) Stride 1 for an input image of size 7x7



(b) Stride 2 for an input image of size 7x7

Figure 2.3: The figure depicts the stride operation in a CNN. The filter moves based on the size of the stride for each neuron connection in the next layer.

next operation in a max-pooling operation. There are other pooling operations like the average of the rectangular neighborhood, L^2 norm of the rectangular neighborhood that could be used instead of the max-pooling operation based on the task at hand.

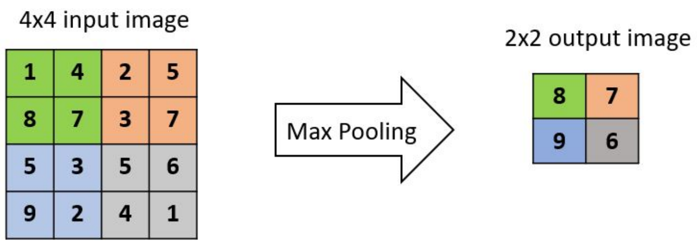


Figure 2.4: The max pooling operation executed in the 4x4 input image with a 2x2 filter and a stride of 2

The main advantage of using a pooling operation is its ability to remain invariant to small translation in the input [11]. The pooling operations are helpful in applications where the presence of the feature is more relevant than the location of these features. In general, the pooling operations increases the computational efficiency and the reduce the memory requirement for computation.

Receptive Fields

In a neural network, the receptive field (*RF*) or the field of view of a neuron in a layer corresponds to the number of neurons connected to it from the previous layer. An effective receptive field (*ERF*) is an area in the input image that could influence the activation of a neuron in CNN [16]. In a CNN, the RF is simply the filter size used in a particular layer, but the ERF footprints the extent of the input image that influences the value of the neuron in a given layer, as depicted in Fig 2.5. It is important to note that any feature outside the receptive field in the input image will not affect the value of the neuron. Therefore, it is important to control the ERF such that all the relevant region in the input image is covered [17].

There are many methods to increase the receptive field of a model. One is to increase the model depth, thereby increasing the receptive field. The other method is by using sub-sampling and dilation methods that increase the receptive field of the model [17]. The effect of an increase in a receptive field has made the CNNs more accurate. [18] introduces receptive field block in an object detection framework and has noticed an increase in accuracy of the frameworks. Dilated convolution neural network is one other way to increase the receptive field. It was introduced in [19] for semantic segmentation task where the state-of-the-art model was established on the Pascal-VOC dataset. The performance of the model in [19] corresponds to the increase in receptive field where important information is not lost for making dense pixel predictions. However, it is important to note that not all the regions enclosed in

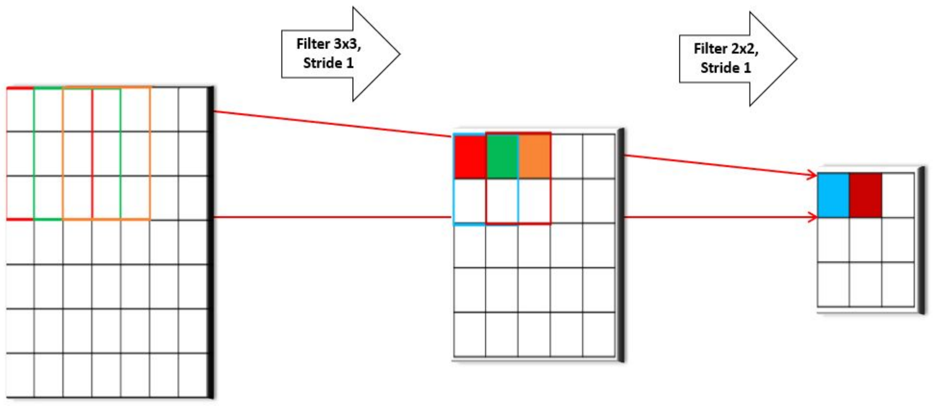


Figure 2.5: The effective receptive field for a particular neuron in the output layer

the receptive field contribute equally to the model's output. [17] investigated the effect of the receptive field in the CNNs and found out that the model's results are calculated based on only a fraction of the receptive field rather than the theoretical receptive field. The fraction of output of the receptive field is a Gaussian distribution, i.e., the central region within the receptive field contributes more to the activation of the neurons.

2.2. Hour-glass structure

An hour-glass network uses a series of convolutions to capture information across all the scales of images and produces a pixel-wise output [20, 21]. The term hour-glass is used to describe these networks because, the steps of pooling and subsequent upsampling to get the final output resembles an hour-glass structure. The hour-glass networks pools down the image into a very low resolution and then upsamples it to get the pixelwise prediction. The hour-glass structure is depicted in Fig 2.6

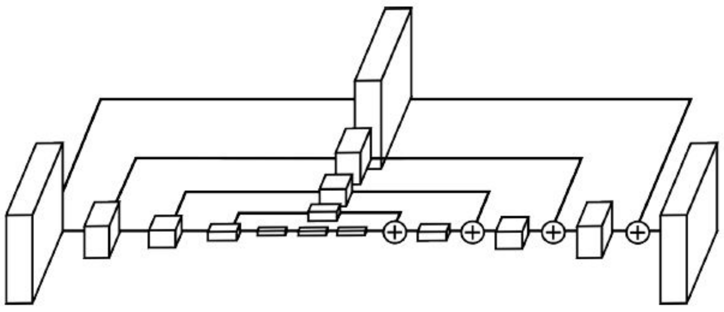


Figure 2.6: A representation of hourglass structure [20]

The hour-glass structure could use a normal upsampling function and skip connections to get the final output (as depicted in Fig 2.6). It is also possible to use deconvolutional layers to learn the spatial information lost in the encoding stage. However, in a hour-glass structure, the encoder and decoder layers must be symmetrical. The backbone used in this thesis is ResNet architecture and is explained in the next section.

2.2.1. ResNet Architecture

Microsoft introduced the ResNet architecture [22] and was the state-of-the-art for image classification with an accuracy of 96.4%. It is a widely used backbone architecture used in many applications like segmentation, object detection, etc. The ResNet architecture was introduced to overcome the problem faced due to the increased depth in the convolutional neural network. Deep models faced the problem of vanishing/exploding gradients which made it difficult to converge the model from the beginning. The ResNet overcomes this problem by introducing residual block with introduces shortcut connections, as depicted in Fig 2.7. The shortcut connections add identity mapping of the input to the layer's output without adding additional parameters. The use of a shortcut connection in a ResNet was able to overcome the gradient problem.

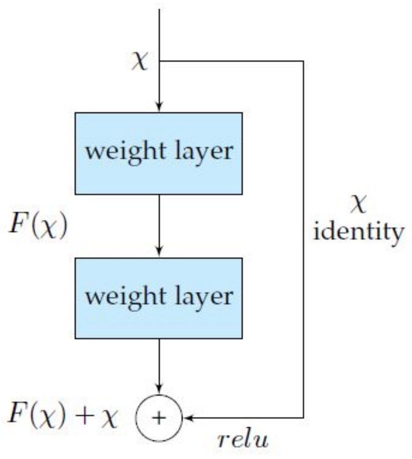
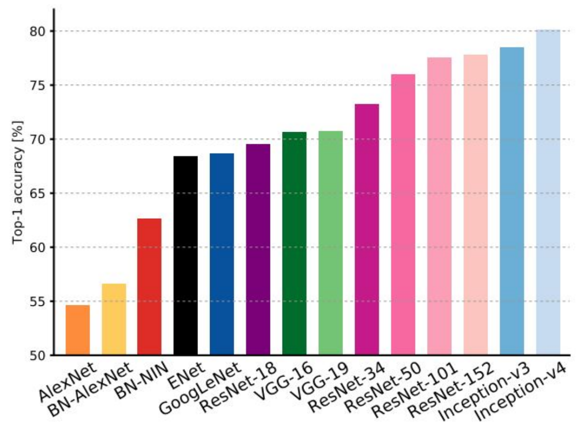
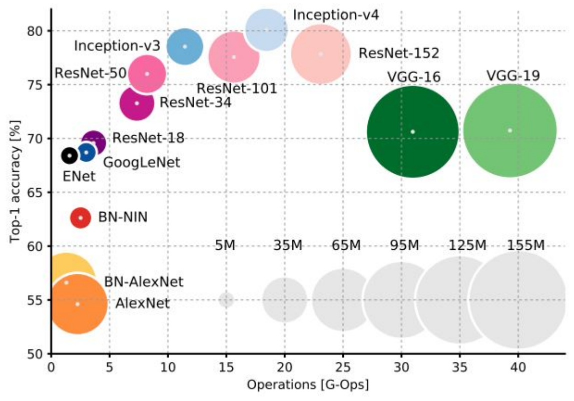


Figure 2.7: The Residual block used in ResNet architecture [22]

Many ResNet baseline architectures are introduced based on depth. Some of the examples ResNet architecture is ResNet 34, ResNet 50, ResNet 101, ResNet 154. Increasing the depth of the ResNet increases the accuracy of the model. However, by increasing the depth, more parameters are introduced into the model [23]. Hence, the working memory for the model is also increased. Therefore, it is important to choose a ResNet backbone considering the memory and the task available at hand. The accuracy of the various ResNet architecture is depicted in Fig 2.8a and it performance compared to the size of parameters is depicted in Fig 2.8b.



(a) The accuracy of the various ResNet models are compared with each other.



(b) The accuracy of the model with respect to the operation memory. The size of the blob corresponds to the number of parameter of the model.

Figure 2.8: The comparison of ResNet architecture with other latest architectures based on the performance. [23]

2.3. Object pose/position estimation for robotic and sequential manipulation

Deep learning methods, especially convolutional neural network has performed well in processing images and videos. This substantial improvement could be attributed to the availability of more computational resources and training data, in addition to the introduction of new techniques [24]. In robotic manipulation, CNNs are used for processing and extracting the pose/position of the object in the image. These pose/position extracted from the image could be used as the high-level goal given by the high-level controller. This section will review some of the works that use vision-based techniques for robotic and sequential manipulation tasks.

2.3.1. Learning manipulation tasks using object-centric model

An object-centric model requires object-specific data to train. They are generally used in pose estimators. The pose estimators extracts the position and the orientation of the object in 3D coordinates. The 3D location of the objects is then given into the planner to manipulate the objects in the image. In a conventional pose estimation method, the common vision-based technique is first used to extract the 2D location of the object in the image. The 2D locations are converted to a 3D position in the world coordinate using pose estimation methodologies like correspondence methods [25], template-based methods [26], etc as depicted in Fig 2.9 [27].

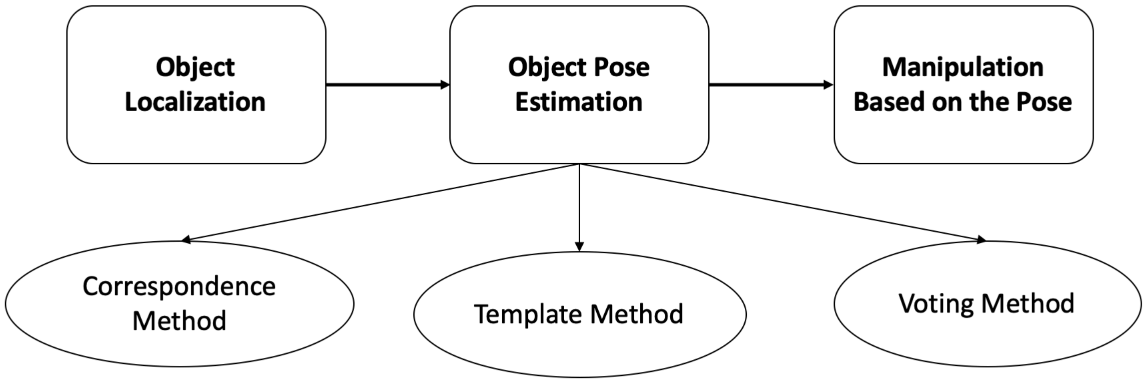


Figure 2.9: The pipeline used to manipulating objects based on vision-based techniques. The object is first localised using a vision-based techniques to get the 2D location in the image. The 2D location is then converted to 3D coordinate using object pose estimation methods [27]

Many methods in a pose estimation algorithm uses a vision-based technique like object detection and segmentation algorithm for extracting the 2D position of the object in the image [28–31]. The object detection and segmentation algorithms are explained in the Appendix A.2. It is also possible to directly extract the 3D point using deep learning bypassing the pose estimation methodologies. [32] uses a PointNet [33] to directly estimate the required 3D coordinates of the manipulation. [34] estimates the 3D coordinates by using a RGB stereo images. The depth is estimated by using triangulation method. There are methods that were able to have a category-level manipulation, i.e., predicting various 3D points in a single object. These methods are especially used in estimating the robot gripper positions [35]. [9] used a object detection method for solving a sequential task. The method used a seq-to-seq model [36] for learning symbols that corresponds to the policy that is to be executed. The object detection algorithm will detect the object and then based on the list of output from the seq-to-seq model, a sequential interaction will be completed.

While all these methods are high accurate in a structured environment, they require either the complete structure of the objects (through 3D models or point clouds) or a large amount of task-specific training data. Therefore, these methods could be less flexible when encountering unseen objects. In addition to it, these methods could not achieve the said accuracy with fewer demonstrations/data.

2.3.2. Learning manipulation tasks without using object-centric model

To overcome the disadvantage faced by object-centric models discussed in section 2.3.1, methods were introduced that decoupled networks from requiring object specific models or larger amount of data [37–39]. [37] introduced the Deep spatial autoencoders, an unsupervised technique for estimating the spatial manipulation points in the image. This method is claimed to be data-efficient and is discussed in detail. [38] introduced a general matching function that could match the pick and their corresponding place location with images. The methods implements a Siamese network with shared weight to generate candidate place locations and orientations. Finally, the method uses a planner to generate the required pick and place locations. [40] used a contrastive loss function to match the pick and place locations.However, [39] introduced the transporter networks that work on a similar concept but are more effective(use fewer demonstrations) than [38, 40]. Hence this method too will be discussed in detail in the coming sections.

For the literature, two main networks were considered to be able to learn sequential pick and place using fewer demonstrations. These two architecture are explained in the next section.

Deep spatial auto-encoders (DSAE) [37]

Deep spatial auto-encoder(DSAEs) is an unsupervised learning technique that is used for learning vision-based manipulation [37]. The DSAEs learns the state of the environment by learning feature points that corresponds to the object's position in the image. The architecture of a DSAE is depicted in Fig 2.10.

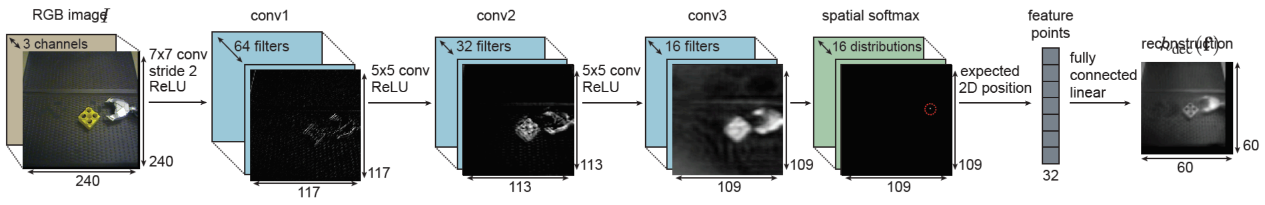


Figure 2.10: The architecture of a DSAE. [37]

Architecture

A DSAE has an encoder-decoder structure to extract the feature points. In encoder-decoder structure the image is first downsampled into a lower-dimensional vector. These low-dimensional vectors, also known as latent vectors, are usually dense and represent the global feature of the high-dimensional input. Such lower dimensional vectors represent the state of the environment and could be used in learning based control algorithms like reinforcement learning or imitation learning [41]. However in a DSAE, the low-dimensional vectors encode the spatial location of the dense latent vectors. The spatial locations are encoded by using a spatial softmax which extracts the spatial location of the pixels, as represented by the formula given in Eq 2.10.

$$s_{cij} = \frac{e^{a_{cij}/\alpha}}{\sum_{i,j} e^{a_{ci'j'}/\alpha}} \tag{2.10}$$

where,

$$\begin{aligned} a_{cij} &= max(0, z_{cij}) \\ c &= \text{channels} \\ (i,j) &= \text{pixels location in 2D} \end{aligned}$$

The encoder part of the architecture is a standard 3-layer convolutional layer that extracts the lower-dimensional latent space. The upsampling is done using a simple nearest neighbor method. A simple 64X64 image is reconstructed. Since the architecture has minimal convolutional layers, the number of parameters to be learnt are lesser, and hence the DSAEs are data-efficient.

The spatial locations are unconstrained for a static environment. In order to constraint the generation of feature points, a penalty function is added to predict feature points that have a small change in velocity. This additional penalty function g_{slow} is added to the overall loss function. The loss function is depicted in Eq 2.11.

$$\mathcal{L}_{\mathrm{DSAE}} = \sum_{t,k} \left\| I_{\mathrm{downsamp},k,t} - h_{\mathrm{dec}}\left(\mathbf{f}_{k,t}\right)\right\|_2^2 + g_{\mathrm{slow}}\left(\mathbf{f}_{k,t}\right) \tag{2.11}$$

Here, $I_{downsamp}$ depicts the input image (downsampled to 64X64), h_{dec} is the reconstructed image using the feature points $f_{k,t}$ at time t. The g_{slow} is the penalty function computed using $g_{\text{slow}}\left(\mathbf{f}_t\right) = \|(\mathbf{f}_{t+1} - \mathbf{f}_t) - (\mathbf{f}_t - \mathbf{f}_{t-1})\|_2^2$.

Eventhough, the use of a penalty function constrains the model to generate spatial point on the relevant objects in the image, it is difficult to teach the network what relevant feature points to concentrate for a given time. Sometimes, it is also possible to have noisy feature points or the absence of feature points due to occlusion. To handle these problem, the concept of feature presence and feature pruning was introduced.

Feature presence is a methodology that is used to predict a feature point, in frames the particular feature point is actually absent due to occlusion. The presence of a feature is determined by the softmax activation. A kalman filter is trained on feature points with softmax activation greater than a threshold (β=0.95). This kalman filter is then used to predict the feature points when it is actually absent in a given frame. **Feature pruning** was used to remove the noisy feature points from the entire sequence of the demonstration. These noiseless feature points of the entire sequence is then used to train a simple linear Gaussian controller that is used to manipulate objects.

It is important to note than the prediction of the vision network is not fully independent. Eventhough, the DSAE used were able to output spatial locations data-efficiently, they were not able to give the noiseless feature points without the inclusion of feature presence and pruning methodologies. The individual performance of the DSAE is not evaluated in the paper. It could be possible that the predictions of the DSAE could be overlapped with the prediction from the kalman filters. Therefore, the DSAE were not considered for this thesis.

Transporter Network [39]

Google introduced transporter Networks for vision-based manipulation tasks and is the basis for our thesis. They are an end-to-end approach that maps image pixels to the required action. The significance of a transporter network is that they perform on unseen objects with fewer demonstrations by obtaining more than 90% success rate on tabletop manipulation tasks.

Transporter networks learn to pick an action based on visual observation o_t as depicted in Eq (2.12).

$$f\left(\mathbf{o}_t\right) \rightarrow \mathbf{a}_t = \left(\mathcal{T}_{\text{pick}}, \mathcal{T}_{\text{place}}\right) \in \mathcal{A} \tag{2.12}$$

where,

$$\mathcal{T}_{\text{pick}} = \text{the location of pick}$$
$$\mathcal{T}_{\text{place}} = \text{the location of place}$$

Here the action A is the collection of both the pick and place poses of the robot. The actions are considered as the end-effector position of the robot in 3D space, and a simple motion primitive is applied to move the robot from T_{pick} to T_{place}. To correspond each pixel in the observation o_t to action, an orthogonal projection is applied. RGB images and depth maps are extracted from the camera in this projection, and the 3D point cloud is calculated. The 3D point cloud represents the point in world coordinates. These point clouds are then orthogonally projected to get a table-top view of the scene. The resulting observation is, therefore, occlusion-free and could correlate pixels to 3D actions.

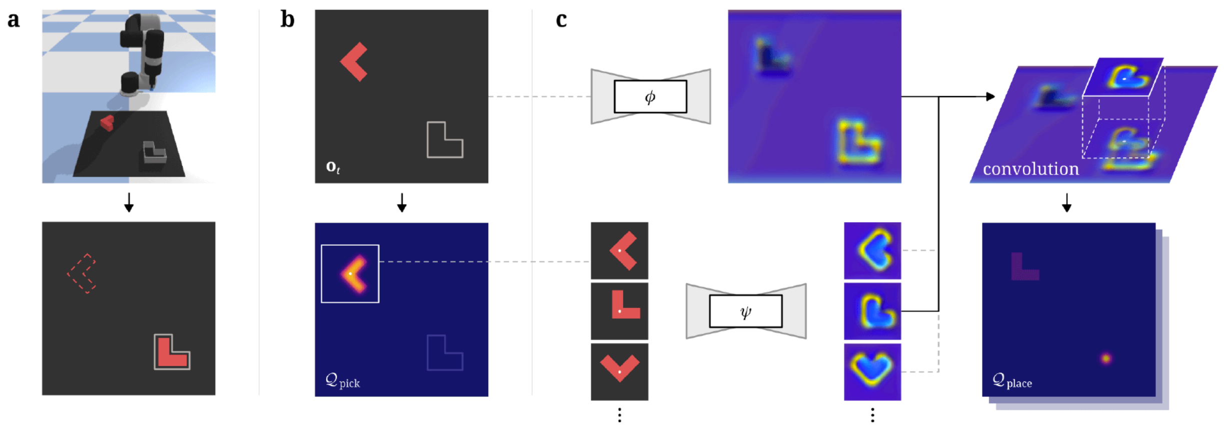


Figure 2.11: The implementation of Transporter Network [39]

To transport an object from T_{pick} to T_{place} location, the model must learn to extract these two points from the observations, as depicted in Fig 2.11. Therefore, the overall transporting is decomposed into two tasks namely,

1. Learning to pick
2. Learning to place based on the pick

Learning to pick: The transporter network uses a fully convolutional network usually used in image segmentation tasks for predicting the pick location from observation. For this, the transporter uses a ResNet hour-glass structure (explained in detail in Appendix 2.2) to predict the distribution of successful picks for all pixels in the observation o_t by using an action-value function. The pick location is predicted using Eq (2.13).

$$\mathcal{T}_{\text{pick}} = \underset{(u,v)}{\operatorname{argmax}}\, softmax\left(\mathcal{Q}_{\text{pick}}\left((u,v) \mid \mathbf{o}_t\right)\right) \tag{2.13}$$

where,

$$\mathcal{Q}_{\text{pick}} = \text{action-value function}$$
$$u, v = \text{pixel location in 2D}$$

For having the a pick location, a image-wide softmax is implemented to the dense-pixel wise prediction from the ResNet hour-glass structure and the location of the maximum probability in the image is extracted. The ResNet architecture used is a 8-stride architecture (3 2-stride convolution in the encoder) as a balance between the receptive fields and the loss of resolution in the latent representation of the hour-glass structure.

Learning to place based on the pick: This part of the transporter network is the unique part where the place location is predicted based on the predicted pick location. For correct prediction, the objects are assumed to have a spatially consistent visual representation, i.e., the object's appearance remains similar across various camera views.

The visual observation o_t is first converted to a dense pixel-wise prediction using a separate network in this part of the transporter network. This prediction represents the candidate place poses τ based on the function $\phi\left(\mathbf{o}_t\right)$. A crop is taken around the pick location $\mathcal{T}_{\text{pick}}$ on the visual observation o_t. These crops are passed into another network to get a dense pixel-wise prediction represented by the function $\psi\left(\mathbf{o}_t\left[\mathcal{T}_{\text{pick}}\right]\right)$. The uniqueness of the transporter network is to find the best placement based on the highest feature correlation using convolution function. Both the models used for generating the dense-pixel wise mapping uses the same ResNet hourglass structure as the picking model but without the final image-wide softmax activation. This methodology is expressed in Eq (2.14).

$$\mathcal{Q}_{\text{place}}\left(\tau \mid \mathbf{o}_t, \mathcal{T}_{\text{pick}}\right) = \psi\left(\mathbf{o}_t\left[\mathcal{T}_{\text{pick}}\right]\right) * \phi\left(\mathbf{o}_t\right)[\tau] \tag{2.14}$$

After the correlation, the image-wide softmax is calculated, and the placing location is calculated based on Eq (2.15).

$$\mathcal{T}_{\text{place}} = \underset{\{\tau_i\}}{\operatorname{argmax}}\ softmax\left(\mathcal{Q}_{\text{place}}\left(\tau \mid \mathbf{o}_t, \mathcal{T}_{\text{pick}}\right)\right) \tag{2.15}$$

It is also possible to learn planar rotation using a transporter network. For this, the rotation is discretized into k bins, and visual observation is rotated with an angle corresponding to each discretized bin. During the prediction of the place point, the rotation value corresponding to the bin with maximum correlation is considered the orientation for planar rotation. *The transporter networks are also performs well on sequential tasks*. The networks can predict the actions based on the position of the objects in the environment. This feature is made possible in the model by increasing the receptive field of the model for covering most of the visual observation(as discussed in section 2.1.2). The receptive layer in the transporter network is based on the trade-off with the latent representation in the hour-glass structure.

This section discussed the detailed working of transporter network. The transporter network was able to perform well in sequential tasks by increasing the receptive field. Increasing the receptive field contributes to the transporter network's prediction conditioned on the objects present within the receptive area. However, the contributions of objects within the receptive field to the output are not equal and are in the form of a Gaussian distribution [17]. Moreover, in some cases, the object can lie outside the receptive field. In this case, the model's prediction will not be based on all the objects in the scene. Hence it important for the model to be independent from the size of receptive field to solve a sequential task. Since the transporter networks are already data-efficient and performs well on unseen objects, this thesis considers this method to solve sequential tasks. The thesis will mainly explore possibilities to overcome the limitations faced by the current transporter networks.

2.4. Metrics

Metrics are used to evaluate the performance of a deep learning model. Usually, in a manipulation task, a robot performs an action based on the output from the vision network. The performance of the model is then evaluated based on the success of these actions [37, 39]. In this thesis, only the computer vision network is evaluated. Hence a separate metric is required to evaluate the performance of the model.

2.4.1. Object Keypoint Similarity (OKS Metrics)

The OKS Metric is an often-used metric to evaluate the performance of a human pose estimation framework [42]. The main aim of the OKS metric in this thesis, is to measure the closeness of the predicted pick location of the network to the ground truth.

To calculate an OKS metric value, a keypoint similarity (ks) is first calculated based on Eq (2.16).

$$ks\left(gt_i, p_i\right) = e^{-\left(\frac{\|gt_i - p_i\|_2^2}{2\sigma_{threshold}^2}\right)} \tag{2.16}$$

The distance between the ground truth (gt) and the prediction (p) is first calculated and normalized to a Gaussian distribution. The standard deviation of the Gaussian distribution is pre-defined based on the tolerance of the accuracy measure represented by $\sigma_{threshold}$. If the evaluation needs to be strict, a lower value of $\sigma_{threshold}$ is used and vice-versa. After normalizing the values to a Gaussian distribution, the OKS metric is calculated based on the visibility score. This equation is expressed in Eq (2.17).

$$OKS\,(gt_i, p_i) = \frac{\sum_i ks\,(gt_i, p_i)\,\delta\,(v_i > 0)}{\sum_i \delta\,(v_i > 0)} \tag{2.17}$$

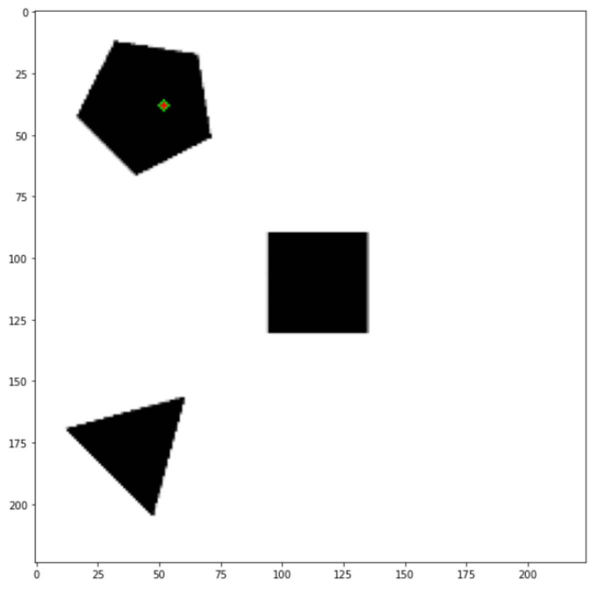
In this thesis, all the key points are assumed to be 100% visible or visibility score of 1 ($\delta_i = 1$). The number of keypoints per class is also considered one ($i = 1$) because each object of interest in an image has only one point used for manipulation. Hence Eq (2.17) is simplified to Eq (2.18)

$$OKS\,(gt_i, p_i) = ks\,(gt_i, p_i) \tag{2.18}$$

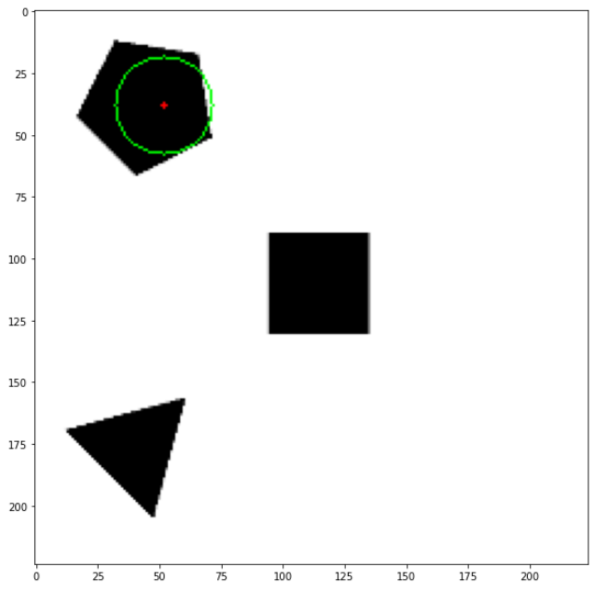
There are two criteria that could be set for calculating the OKS Metrics. Firstly by setting the standard deviation ($\sigma_{threshold}$) for an OKS Metrics, the point with distance more than the standard deviation, from the ground-truth point has an OKS Metrics of 0. Secondly, after normalising to a Gaussian distribution, the points near the tail of the Gaussian distribution have low probability. To eliminate this an additional threshold of 0.5 is chosen to consider predictions closer to the center of the Gaussian distribution . This threshold is represented as $\gamma_{threshold}$. The number of points that satisfy both the criteria is then averaged with the total number of test data points to calculate the accuracy of the network as in Eq (2.19).

$$Accuracy = \frac{OKS\,(gt_i, p_i) > \gamma_{threshold}}{\text{Total number of predictions made}} \tag{2.19}$$

The example of the OKS Metric is depicted in Fig 2.12.



(a) The metrics with a $\sigma_{threshold}$ of 2 pixels



(b) The metrics with a $\sigma_{threshold}$ of 20 pixels

Figure 2.12: The green circle represents the standard deviation criteria in an OKS Metric. If the predicted point falls inside the green circle, a probability value is generated. This probability value should be more than 0.5 to be considered as a correct prediction in this thesis.

It is essential to note that the error in predicting the pick point for a sequence will directly affect manipulating the object in the actual scene. Therefore, it is important to note the relation between the change in the robot's end-effector position to the change in spatial point location in an image through camera-robot calibration. Since the dataset used is a synthetic dataset and a pre-recorded Mujoco dataset, this camera-robot calibration information is not available. Therefore, the results will be presented for a very strict value of $\sigma_{threshold}$ of 2 pixels (as depicted in Fig 2.12a) and a lenient value of $\sigma_{threshold}$ of 20 pixels (as depicted in Fig 2.12b). The result will then be presented for $\gamma_{threshold}$ of 0.5.

2.4.2. Classification Accuracy

Classification accuracy is the usual measure of accuracy. It is defined as the ratio of the number of samples classified correctly by the network to the total number of samples as shown in Eq (2.20).

$$\text{Classification Accuracy} = \frac{\text{Number of Correct predictions}}{\text{Total number of predictions made}} \tag{2.20}$$

The classification accuracy is used to evaluate the ability of SEA Net to predict what spatial point of the object in the environment to choose given a particular configuration.

3 Multi-output Pick Network

In this chapter the multi-output pick network is introduced, which modifies the pick network of a transporter network. The chapter first explains what a multi-output pick network is and describes how it is implemented.

The transporter network is explained briefly in Section 2.3.2. The pick network in a transporter network predicts the pick location in the image, while the place network gives the place location in the image. It should be noted that the place location depends on the prediction of pick location. Hence, the effect of receptive fields on the pick location is first studied and a solution is proposed. The proposed solution could replace the existing pick network in the transporter network without any problem.

In this section, the multi-modality of the pick network is studied. This means that the pick network produces multi-modal outputs in situation where a single solution exists. A multi-modal output means that two or more probability peaks are predicted by the model, as depicted in Fig 3.1. To understand this situation more clearly, assume there are two demonstrations: 1. A demonstration to pick and place a cube, 2. A demonstration to pick and place a sphere. Let the pick model be trained with both the cube and sphere demonstrations. Since the pick network is exposed to both cube and sphere during training, it exhibits a high probability in location of cube and sphere (or objects similar to cube and sphere). Therefore, the network will have a small bias toward choosing one of the probabilities instead of the other. In the sequential task considered in this thesis, all objects to be manipulated are present in the scene at every instant. Therefore, the pick model can give multi-modal outputs for objects with different shapes and sizes used in sequential tasks. Hence, in this section a variation of the pick network is introduced where instead of generating one N-modal output, the network produces N-unimodal output with the same receptive field.

3.1. Architecture

To overcome the multi-modal constraint of the pick model, the model needs to learn to produce uni-modal outputs. This should be possible without changing the receptive fields of the network because this thesis aims to break the dependence of the network on the receptive field size. This is made possible with the introduction of a multi-output pick network. The multi-output pick network calculates k action-valued functions that corresponds to k pick locations, where k is the number of objects to be manipulated in the sequential task. Introducing multi-output in the pick network will force the model to

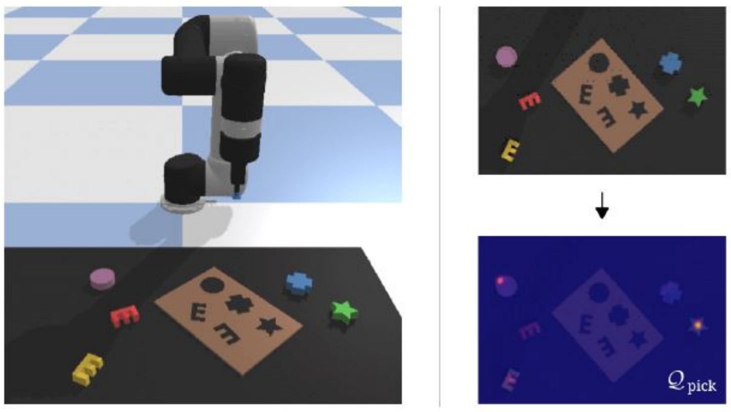


Figure 3.1: The multi-modal output of the transporter network. The image in the left shows the scene with more than one similar object the model has seen during training. The image in the right shows the multi-modal output of the pick model [39]

learn uni-modal outputs corresponding to each object in the sequence. These uni-modal outputs will help the network to clearly choose the highest probability.

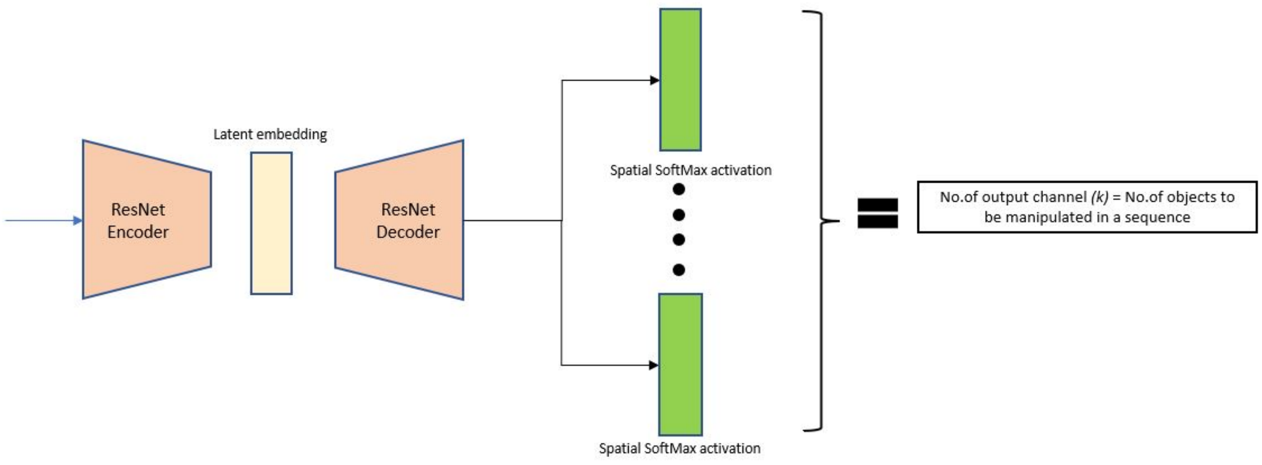

Latent embedding
ResNet Encoder
ResNet Decoder
Spatial SoftMax activation
Spatial SoftMax activation
No.of output channel (k) = No.of objects to be manipulated in a sequence

Figure 3.2: The multi-output pick network with multiple output channels. Each channels predicts the output of each objects in the scene separately.

$$\mathcal{T}^{k}_{\text{pick}} = \underset{(u,v)}{\text{argmax}}\ softmax\left(\mathcal{Q}^{k}_{\text{pick}}\ ((u,v) \mid \mathbf{o}_t)\right) \tag{3.1}$$

where,

$$\begin{aligned} \mathcal{Q}_{\text{pick}} &= \text{action-value function} \\ u, v &= \text{pixel location in 2D} \\ k &= 1,2,\ldots,p \text{ where p = number of objects to manipulate} \end{aligned}$$

The multi-output pick network uses the same hour-glass structure (explained in detail in Section 2.2). A ResNet 43 layer architecture is used with 12 residual blocks with 8-stride (3 2-stride convolutions in the encoder and 3 upsampling in the decoder), with an output consisting of multiple channels, which correspond to the number of objects to be manipulated in a sequential task, as depicted in Fig 3.2. The output from the ResNet is then used to calculate a image-wide softmax layer for each channel.

3.2. Loss Function

The cross-entropy loss function can be employed to compute the loss between the prediction and the ground-truth. A cross-entropy is used to compare one probability distribution with another probability distribution in a multi-class problem [43]. In a multi-output pick network, each pixel is considered as a separate class. The cross-entropy loss is represented in Eq 3.2

$$\mathcal{L}_{\mathrm{CE}} = -\sum_{i=1}^{n} gt_i \log(p_i)\,, \text{ for n classes} \tag{3.2}$$

where gt_i is the ground truth in the form of one-hot tensor, p_i is the probability obtained from the softmax function.

A cross-entropy loss function is used for training. The cross-entropy loss function is used between the softmax output of each channel and their corresponding one-hot pixel. One-hot pixel maps are required because each pixel is considered as a seperate class and the ground-truth for each pixel is required for computing the loss. Since the multi-output pick network predicts one output for each object of interest in the environment, the summation of each channel's loss is taken as the total loss of the hour-glass network.

$$\mathcal{L}_{\mathrm{Multi-output\ pick\ network}} = \sum_{i=1}^{p} \mathcal{L}_{CE}, \text{k = 1,2,... ,p} \tag{3.3}$$

where,

p = number of objects manipulated in the demonstration

3.3. Discussion

A multi-output pick network can predict the spatial location of every individual object in a scene. This was introduced because the pick network of a transporter network gave multi-modal outputs. Multi-modal outputs biases the network to pick one of the two highest probability peaks irrespective of the configuration of the objects in the environment. To make the output a function of the configuration of the environment, every pixel in the output needs to be correlated with as many pixels as possible in the input. If this is not the case, then each pixels will be only dependent on the neighbouring pixels thereby loosing the capability to produce an output as a function of the configuration of the environment. Receptive fields are increased to incorporate as many input pixels as possible to produce an output. However, the pick network already has a large receptive field and it would be impractical to increase the receptive field even more as the increase would correspond to a loss of information in the latent space of the network. The main contribution of the multi-output pick network is to separate the outputs in the final layer without changing the receptive field of the network and to obtain uni-modal outputs. This could eliminate the dependence of the network on receptive field and will give a uni-modal output for the same receptive field. However, the multi-output pick network is still expected to have a multi-modal behavior outputs the final channels when multiple instances of the same object are present in the scene for sequential manipulation. But it could be assumed that randomly picking one of the same objects will still complete the sequential task successfully. To have multiple channels in the output, this thesis assumes that the number of objects required for manipulation in the scene is known beforehand.

4

Experiment and Results - Pick Network and Multi-output Pick Networks

This chapter will explain the preliminary experiments conducted on the pick network of a Transporter Network and multi-output pick network. This chapter will first introduce the dataset required for the experiment and then discuss the results obtained through the experiments.

4.1. Synthetic dataset creation - Dual Target Dataset

The thesis aims to predict high-level goals in the form of spatial location. These spatial locations need to be predicted sequentially. Sequentially predicting the pick point requires the output from the model to be correlated with each pixel in the image. To check this functionality in the pick network of Transporter Network the *Dual Target dataset* was introduced. This is depicted in Fig 4.1. The aim of using this dataset is to see if the models can make the prediction by correlating all the pixels in the input image.

The Dual Target dataset consists of three different objects randomly placed in the image without any overlap. The images display a top-down view of simple objects like squares, circles, pentagons, and triangles. Each image in the Dual Target dataset will consist of either a square or a circle and will always feature a pentagon and a triangle. The idea behind this setup is that the model should predict the spatial location of a pentagon when a square is encountered in the image and the spatial location of a triangle when a circle is encountered given both pentagon and triangle are always present in the image. The target spatial location in the pentagon and triangle is not always in the center of the object. The target locations are placed randomly on the object such that the same configuration could have a different target location on the object.

To train the pick network, for each configuration the corresponding target spatial location is used as the label. For multi-output pick network, for each configuration all the target spatial locations are used as the label.

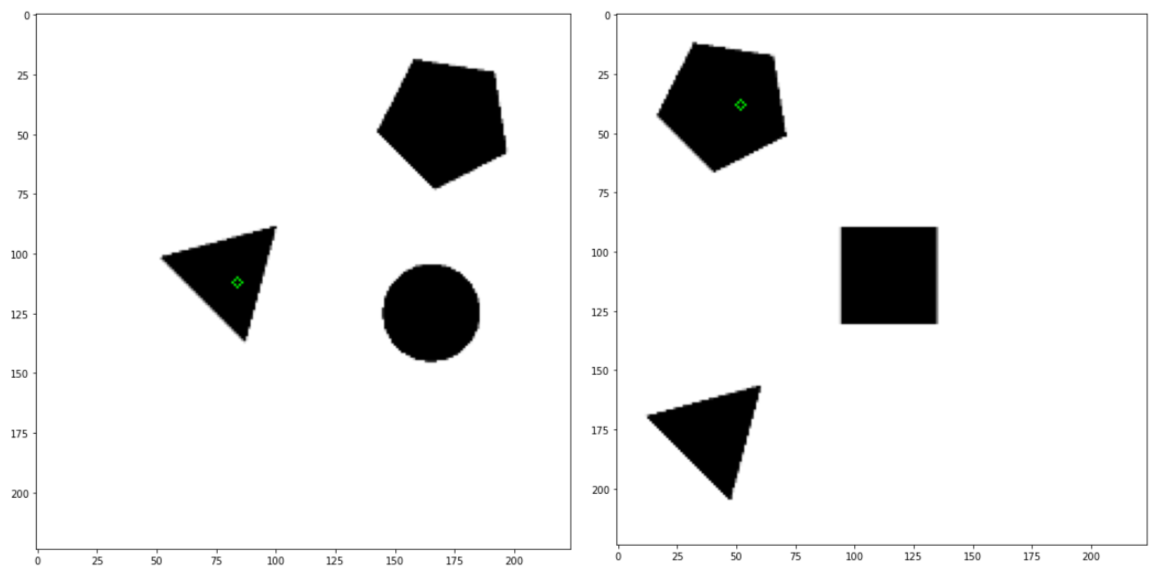


Figure 4.1: The Dual Target dataset has three objects in top-down view placed randomly. The green point indicates the spatial target point for the network to predict. (a)The target location is on triangle when a circle is present. (b)The target location is on pentagon when a square is present

4.2. Pick Network Preliminary Experiments

As discussed in section 2.3.2, a transporter network is composed of two networks. Only the pick network is used for analysing the feasibility of predicting the pick locations in a sequential tasks. The aim of the experiment is to test the performance of the pick network.

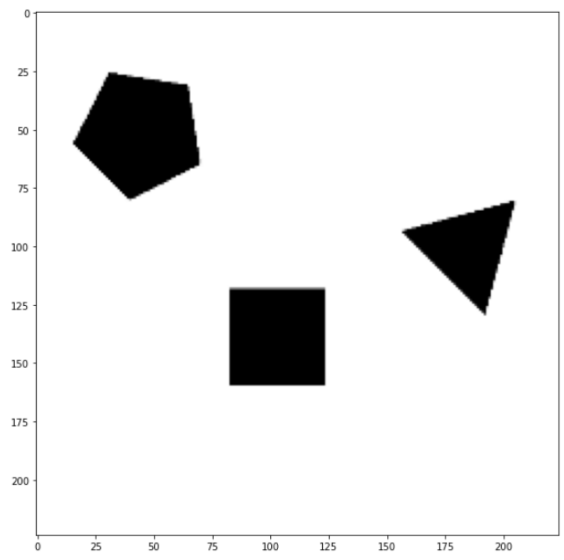
4.2.1. Experimental setup

The pick model of the transporter network is trained with dataset that contain 1000 images similar to the image depicted in Fig 4.1. The model is trained for 75 epochs using Adam optimiser and learning rate of 1e-4.

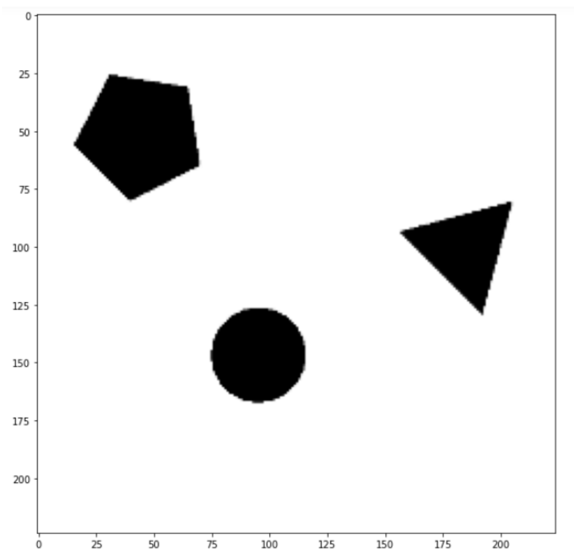
The model is then evaluated on Fig 4.2. Fig 4.2a consist of three blocks namely a square, pentagon and triangle while Fig 4.2b consist of a circle , pentagon and triangle blocks. The aim of the model is to predict the spatial location based on the environment configuration. The aim of the experiment is to check the ability of the transporter network to correlate the pixels of the pentagon and triangle with the pixels of the square or circle in the input image and to predict a output based on this correlation.

4.2.2. Results

Fig 4.3 and 4.4 depicts the prediction of the pick network for the inputs given in Fig 4.2. It is expected that given the larger receptive field, the pick network must provide a single output based on every pixel in the image. However, it could be observed that the pick network produces multi-modal predictions as depicted in Fig 4.3a and 4.4a. The application of softmax activation layer filters out these multi-modalities. However, the model gives the same output irrespective of the configurations. This could be due to the fact that the receptive field of the backbone is not large enough to consider all the information in the environment for prediction. Increasing the receptive field could mean increase in the number of learnable convolutional layers in the backbone, and an increase in the loss of information in the latent space of the network [17]. Hence, it is essential to build a model that is independent from the effect of size of receptive field to solve sequential tasks.

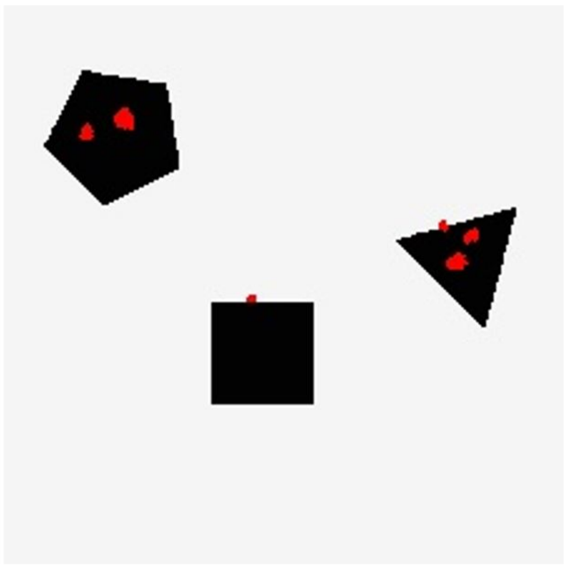


(a)

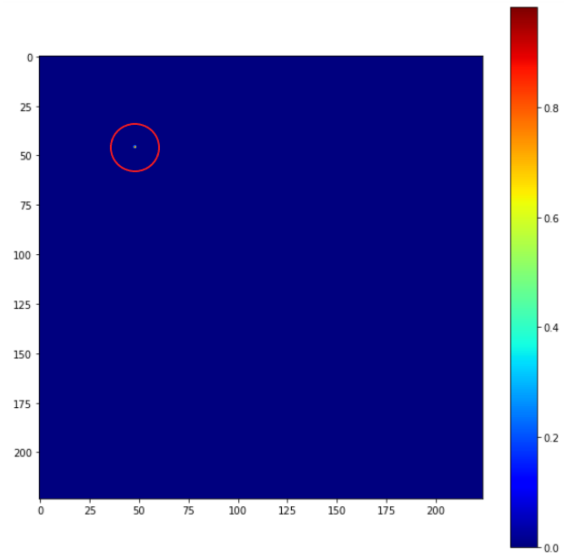


(b)

Figure 4.2: The experiment has two images. The position of the target shapes remains the same. The model needs to predict the output spatial location based on the configuration.



(a) Pixel with high activation before softmax



(b) Softmax activation

Figure 4.3: The activation and softmax prediction for an input depicted in Fig 4.2a. The pixels of maximum activation are depicted in red colour in (a)

To overcome this problem, the network needs to predict uni-modal output for the same receptive field.

4.3. Multi-output Pick Network Preliminary Experiments

As discussed in Section 3, the multi-output pick network was designed to give uni-modal predictions without changing the receptive field size of the network. This section discusses the preliminary experiments carried out using the multi-output pick network.

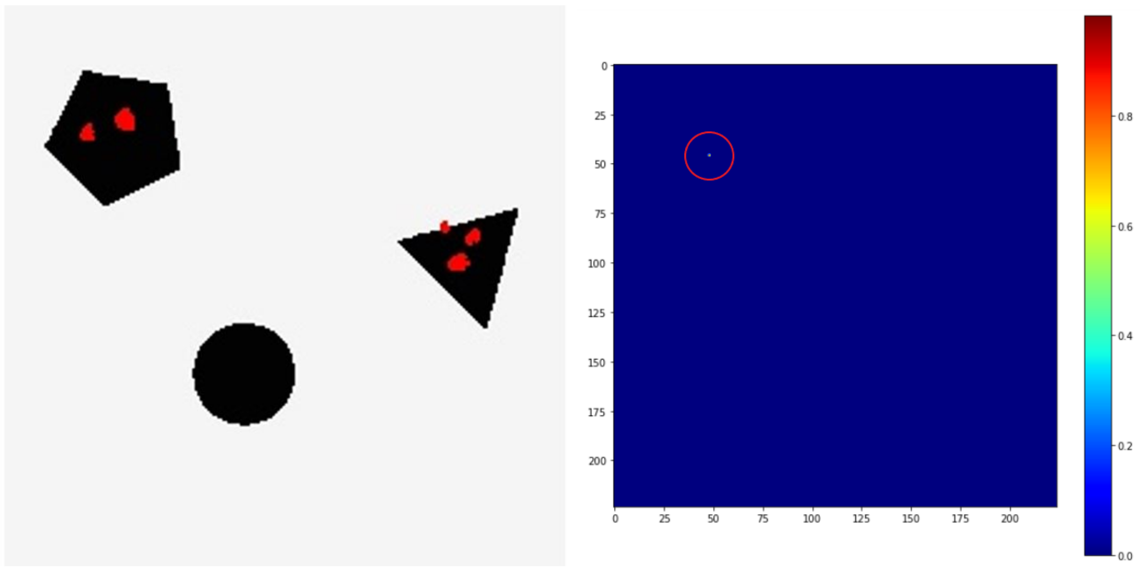

(a) Pixel with high activation before softmax (b) Softmax activation

Figure 4.4: The activation and softmax prediction for an input depicted in Fig 4.2b. The pixels of maximum activation are depicted in red colour in (a)

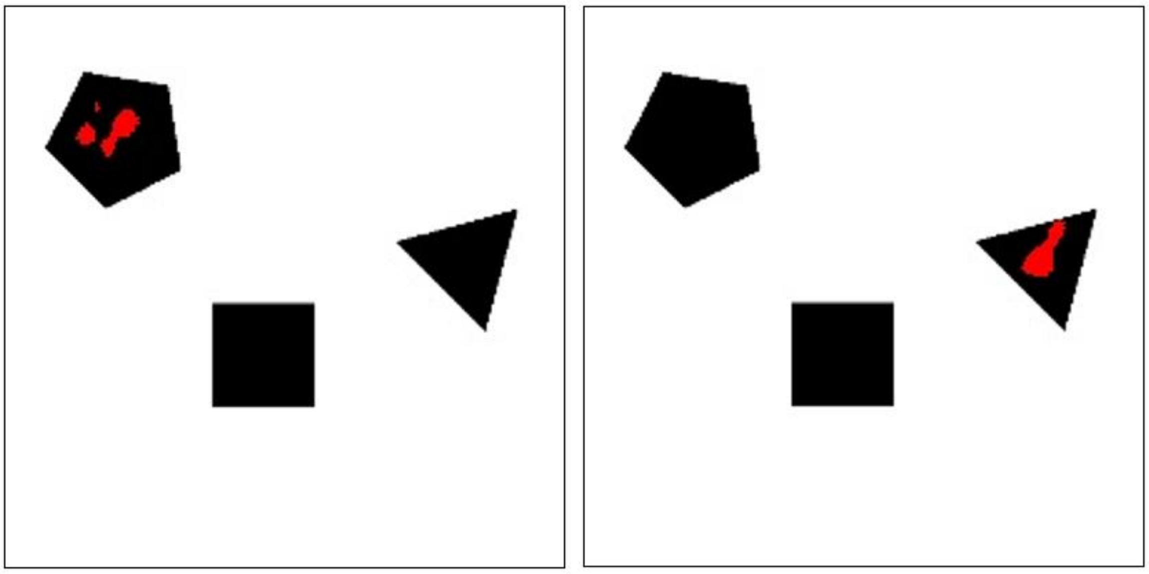
4.3.1. Experimental setup

The multi-output pick network has output channels corresponding to the number of pick locations in the sequence. Since the dataset used to study the performance of the network has two pick locations (as discussed in Section 4.1), the number of output channels for a multi-output pick network is two. The multi-output pick network is trained with the same dataset used in the pick network in the previous section. For each image in the dataset, the spatial location of both the pick points are given to the network for training the model. The model was trained on 1000 images similar to the image depicted in Fig 4.1. The model is trained for 75 epochs using Adam optimiser and learning rate of 1e-4.

4.3.2. Results

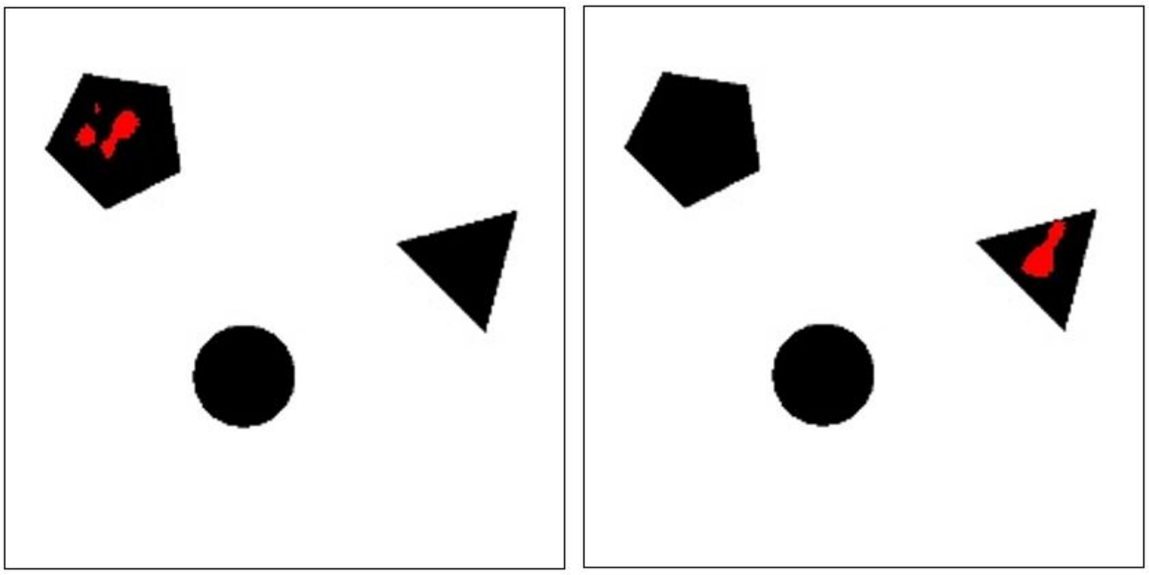
Fig 4.5 and 4.6 presents the output of the multi-output pick network for an input image depicted in Fig 4.2. When compared to the results form the pick network (as depicted in Fig 4.3 and 4.4), the multi-output pick network has uni-modal predictions. These uni-modal predictions are the passed through a image-wide softmax layer to get the pixel with highest probability. The location of this pixel is then used as the target spatial location.

It is important to note that uni-modal outputs were obtained for the same receptive field size by introducing multiple channel. However, for a given time-step, the multi-output pick model will return all the pick location through different channel. Hence, it is important for the network to also chose a particular pick location based on the configuration of the objects in the image. To implement this, the SEquential Attention Networks (SEA Nets) were introduced which will be explained in detail in the next chapter.



(a) Pixel with high activation before softmax on channel 1 (b) Pixel with high activation before softmax on channel 2

Figure 4.5: The prediction of multi-output pick network for an input depicted in Fig 4.2a. The pixels of maximum activation are depicted in red colour in (a)



(a) Pixel with high activation before softmax on channel 1 (b) Pixel with high activation before softmax on channel 2

Figure 4.6: The prediction of multi-output pick network for an input depicted in Fig 4.2b. The pixels of maximum activation are depicted in red colour in (a)

5

SEquential Attention Network (SEA Net)

In this chapter, the SEquential Attention Network (SEA Nets) is introduced, which aids in sequentially predicting the pick points in sequential pick and place tasks. The chapter first explains a SEA Net and describes how it is implemented.

In the Chapter 3, a multi-output pick network was introduced to separate N-modal outputs into N unimodal outputs without changing the receptive field of the model. Separating the outputs will make the network independent of the receptive field size which should be substantially larger to make the prediction based on the configuration. However, the multi-output pick networks, for a given time-step, will provide all the pick locations as discussed in the previous section. Hence, it is important for the model to learn the configuration of the object so that an unimodal solution is obtained for a given image. SEA Net is introduced to learn simultaneously, the spatial/pick points of the objects in the scene and predict which object should be picked at a given time step based on the configuration of the environment, to solve a sequential task. SEA Net has a classification head, which will classify what channel is relevant for a given image.

5.1. Architecture

As discussed in Chapter 3, the multi-output pick network simultaneously predicts the pick location of all relevant objects in the image. The SEA Net introduces an extra classification head into the multi-output pick network. The classification head branches from the latent space of the hourglass network. The prediction of the classification head (κ_{pick}) is based on Eq (5.1).

$$\kappa_{pick} = \operatorname{argmax} p\left(y \mid o_t\right) \tag{5.1}$$

where,

$$\begin{aligned} y &= \text{pick object in the environment} \\ o_t &= \text{Visual observation} \end{aligned}$$

Hence the final output of a SEA Net will be combined with the output of the classification head. This is depicted in Eq (5.2).

$$\mathcal{T}_{\text{pick}} = \underset{(u,v)}{\operatorname{argmax}}\, softmax\left(\mathcal{Q}_{\text{pick}}^{\kappa_{pick}}\left((u,v) \mid \mathbf{o}_t\right)\right) \tag{5.2}$$

where,

$$\begin{aligned}\mathcal{Q}_{\text{pick}} &= \text{action-value function}\\ u,v &= \text{pixel location in 2D}\\ \kappa_{pick} &= \text{prediction of the classification head}\end{aligned}$$

The output of the classification head selects the goal to be reached next in a given time. Based on the output from the classification head, a relevant channel is chosen in the multi-output pick network to get the final pick location.

The overall SEA Net is depicted in Fig 5.1. In addition to the architecture of multi-output pick network, a classification head is added from the latent space for a SEA Net, as shown in Fig 5.1. The classification head consists of a global pooling layer followed by a fully connected layer. The fully connected layer is used to connect all the inputs from one layer to another. By introducing a fully connected layer, the rich global latent features are correlated with each other to output the desired configuration from the image. However, fully connected layers are not translational equivariant and hence are not data-efficient. Translational equivariance will only help in predicting the pick location of the objects which is done using multi-output pick network. Since, the classification head is used to understand which object to be picked next, it is possible that not much data-efficiency is lost because of adding a classification head.

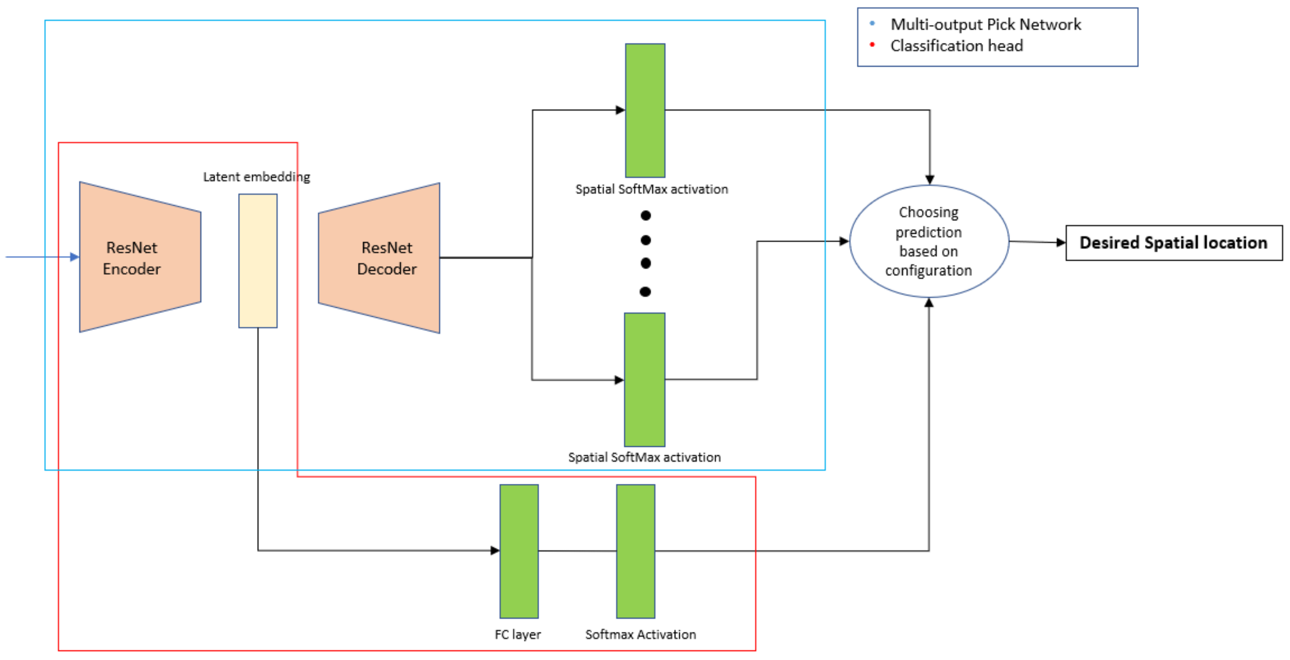

Multi-output Pick Network
Classification head
Latent embedding
ResNet Encoder
ResNet Decoder
Spatial SoftMax activation
Spatial SoftMax activation
Choosing prediction based on configuration
Desired Spatial location
FC layer
Softmax Activation

Figure 5.1: A full model representation of the SEquential Attention Network

5.2. Loss Function

The classification head of the SEA Net uses the cross-entropy loss function because the prediction of the relevant pick object is a multi-class problem. A softmax activation is used to represent the output of the networks as the probability. The ground-truth for the loss function should be a one-hot vector that has the highest probability(1) on the location of the vector that corresponds to the particular class label.

The total loss of the SEA Net is represented in Eq 5.3

$$\mathcal{L}_{\text{SEA Net}} = \mathcal{L}_{\text{Multi-output pick network}} + \lambda * \mathcal{L}_{\text{Classification Network}} \tag{5.3}$$

where,

$$\lambda = \text{weighing constant}$$
$$\mathcal{L}_{\text{Multi-output pick network}} \text{is based on Eq (3.3)}$$

$$\mathcal{L}_{\text{Classification Network}} = -\sum_{i=1}^{n} gt_i \log(p_i)\,, \text{n = 1,2,... ,p} \tag{5.4}$$

$$p = \text{number of objects manipulated in the demonstration}$$
$$gt_i = \text{ground-truth of the object to be manipulated for given configuration}$$
$$p_i = \text{probability of the given class}$$

During each training step, the model receives the ground truth locations in pixel co-ordinate for the objects in the environment and the classification label. For example, if there are three objects in the environment and the sequence is carried on these three objects, all the object's locations and classification label are given as the ground truth label into the model. All the three pixel locations are converted into one-hot tensor. Likewise, the classification label is also converted into one-hot vector. These two labels are then used to train the model.

5.3. Discussion

The SEA Net can predict the relevant pick location required to solve a sequential task. The pick network's performance on sequential task is dependent on the receptive field size of the network. This assumption is broken in the SEA Net as a classification head is additionally introduced along with multi-output pick head, to predict the relevant object to be considered for a given segment of a demonstration. By introducing a classification head in the latent space of the hourglass structure, a output is predicted by correlating the global features, thereby giving a prediction based on the full configuration of the image. However, the classification head is composed of fully connected layers which are translational invariant. Still, this feature of a fully connected layer is simple used to predict the configuration and would not greatly affect the translational variance required to predict the pick location. Hence, it is important to study the data-efficiency of the SEA Nets. The SEA Nets were introduced with the assumption that the number of objects manipulated in the sequential demonstrations is known before training the model. This makes the SEA Net less flexible for online training.

6

Experiments and Results - SEA Net

This chapter describes and discusses the experiments that were carried out to study the performance of the proposed SEquential Attention Networks. For this, it is important to choose/design a well-suited dataset to test the proposed model. This chapter also explains in detail, the dataset used to train and validate the model. It further highlights the contribution for creating new synthetic dataset, and using a simple segmentation algorithm to sequence and extract simulated robot dataset from pre-recorded demonstration.

6.1. Hardware Setup

The network is implemented with Tensorflow and is trained on a CPU with an i7 processor and an NVIDIA RTX 3060 GPU.

6.2. Experiment on SEA Net - Synthetic Dataset

6.2.1. Creation of Synthetic Sequential dataset

The availability of dataset to test the performance of the model on sequential tasks are limited. There is no tailored made sequential dataset that are collected from human demonstrations available for training models to perform sequential tasks. Hence, it is essential to create a well-suited dataset to test the proposed model.

The synthetic sequential dataset consists of simple shapes which are manipulated sequentially. These shapes are randomly placed in the image without any overlap and are manipulated using a simple robot arm represented as the red circle as depicted in Fig 6.1. The performance of the model on the synthetic sequential dataset could prove the ability of the model to predict output based on the configuration of objects in the environment. To train the SEA Net, a dataset $\mathcal{D} = \{\zeta_1, \zeta_2, ..., \zeta_n\}$ of n expert demonstration $\zeta_i = \{(\mathbf{o}_0, \mathbf{a}_0, \mathbf{c}_0), (\mathbf{o}_1, \mathbf{a}_1, \mathbf{c}_1), ...\}$ is required where,
$\mathbf{o}_i$ is the raw pixels from image,
$\mathbf{a}_i = \left(\mathbf{T^{square}}_{pick}, \mathbf{T^{pentagon}}_{pick}, \mathbf{T^{star}}_{pick}, \mathbf{T^{triangle}}_{pick}\right)$

$\mathbf{c}_i$ is the label of the object manipulated at the trajectory.

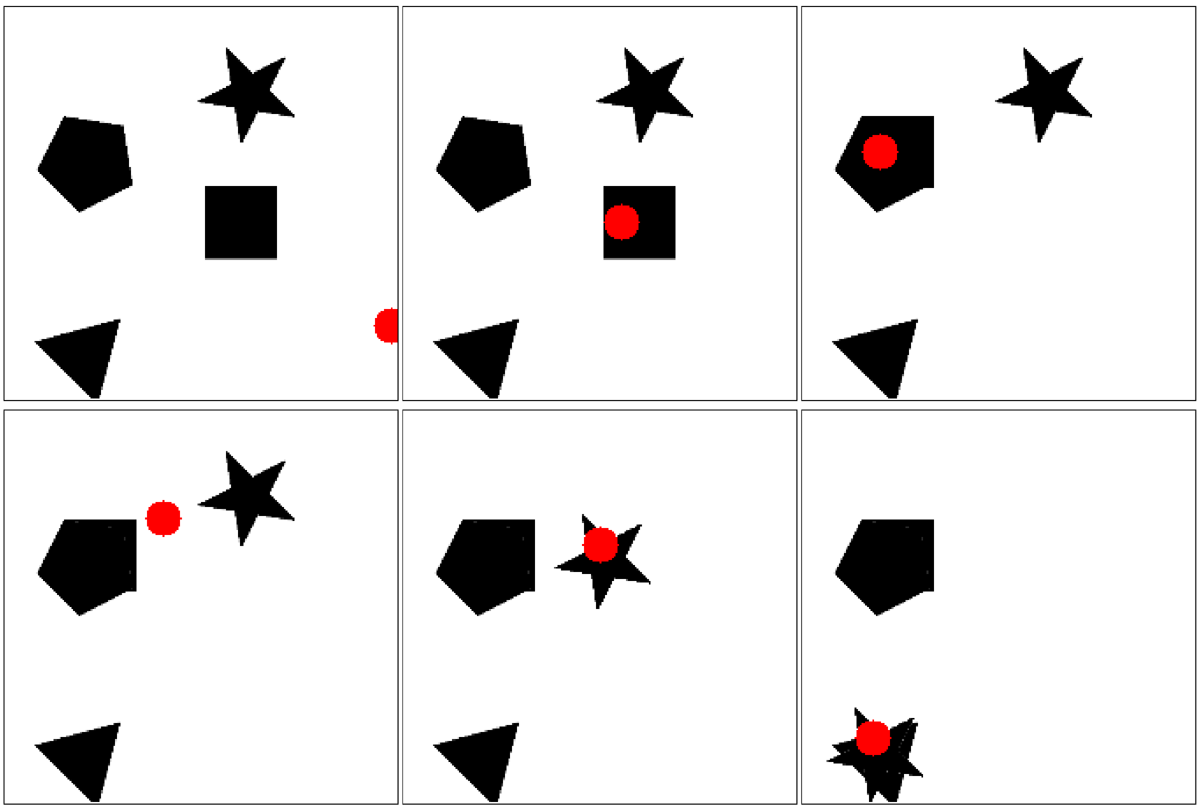


Figure 6.1: The simple sequential dataset created. The end-effector is represented by the red circle. The sequence starts from the top-left corner and end in the bottom right corner.

In Fig 6.1, there are four shapes assumed to be present on the table for manipulation. There are four trajectories in a demonstration.

1. The square is picked by the end-effector. $\mathbf{c}$ = 0
2. The square is placed in goal location (pentagon). $\mathbf{c}$ = 1
3. The star is picked by the end-effector. $\mathbf{c}$ = 2
4. The star is place in goal location (Triangle). $\mathbf{c}$ = 3

6.2.2. Experimental Setup

The network is evaluated on two datasets. The first dataset is a Dual Target dataset (as discussed in Section 4.1) and the second dataset is a full sequential dataset (as discussed in Section 6.2.1). The λ value of the loss function(in Eq (5.3)) is 1. The image is normalized between the values of 0 to 1 to make the processing easier. No pre-trained weights are used in any of the experiments.

Dual Target dataset

As discussed in Section 4.1, each image in a dual-target dataset will have two pick locations that the model is exposed during training. Based on the configuration of the objects in the environment, the model must chose one specific pick location. This experiment could prove the model's robustness to pick an appropriate pick location for a given configuration. In this experiment, the network is trained with 1000 images of resolution 224X224, with an epoch of 75. The learning rate used is 10e-4 with no batch normalization. The network is tested with 500 images.

Sequential dataset

The sequential dataset is already explained in section 6.2.1. This experiment could prove the model's ability to predict the pick locations sequentially based on the configuration (here position of the end-effector of the robot). In this experiment, The SEA Net is trained with 200 demonstration with each demonstration having 18 images. The network is trained for 75 epochs with a learning rate of 10e-4. The network is then tested on 10 demonstrations.

6.2.3. Results

Table 6.1 shows the accuracy of the SEA Net on test images. The classification head has a higher accuracy which depicts the SEA Net's ability to predict the object to be considered for a particular trajectory based on the configuration. However, it should be noted that the accuracy of the SEA Net will depend majorly on the accuracy of the classification head. The reason is that, the output from the classification head is used to choose the appropriate channel in a multi-output pick head of the SEA Net. The overall accuracy is based on Eq (2.19) for the channel selected by the classification head. A higher overall accuracy could mean that the pick points are also predicted near the ground-truth which means that the object could be successfully manipulated.

	Classification Accuracy	Overall accuracy of SEA Net (OKS Metrics)	
		$\sigma_{threshold}$ = 2 pixels	$\sigma_{threshold}$ = 20 pixels
Dual Target Dataset	100%	100%	100%
Sequential Dataset	86.2%	64%	85%

Table 6.1: The accuracy results of SEA Net

The study on Dual-Target dataset shows that the SEA Net chooses the correct pick location based on the configuration of the object in the environment. This property is essential for estimating consecutive pick points, to complete a sequential task.

The second dataset is a sequential task that is depicted in Fig 6.1. Here the SEA Net is expected to predict the correct spatial locations of the object to be manipulated throughout the sequential task based on the position of the end-effector (depicted as the red point). The accuracy of the model is depicted in Table 6.1. The experiment on sequential dataset depicts the SEA Net's ability to predict pick points sequentially. It could be noted that the accuracy of the SEA Net on the sequential dataset is lower than the dual-target dataset. The reason behind this behaviour is because the number of objects to be considered in the sequential task is more than the dual-target dataset. Increasing the number of object to be manipulated could decrease the accuracy of both classification head and multi-output pick head as the SEA Net must learn more output channels and configurations.

The results of the SEA Net is depicted in Fig 6.2.

However, it is possible to increase the accuracy by using more data for training the SEA Net. But this was not explored in the thesis, as the aim was to use a smaller dataset.

6.3. Experiment on SEA Net - Simulation robot dataset

6.3.1. Extracting demonstration from simulators

To test the proposed model, it is necessary to acquire demonstrations of humans performing sequential tasks (eg., sequential pick and place or tower of Hanoi, etc) on the robot. These demonstrations could be provided by humans through teleoperation, kinesthetic operation [44]. However, instead of human demonstrators, it is possible to train policies available in the garage library [45] for performing

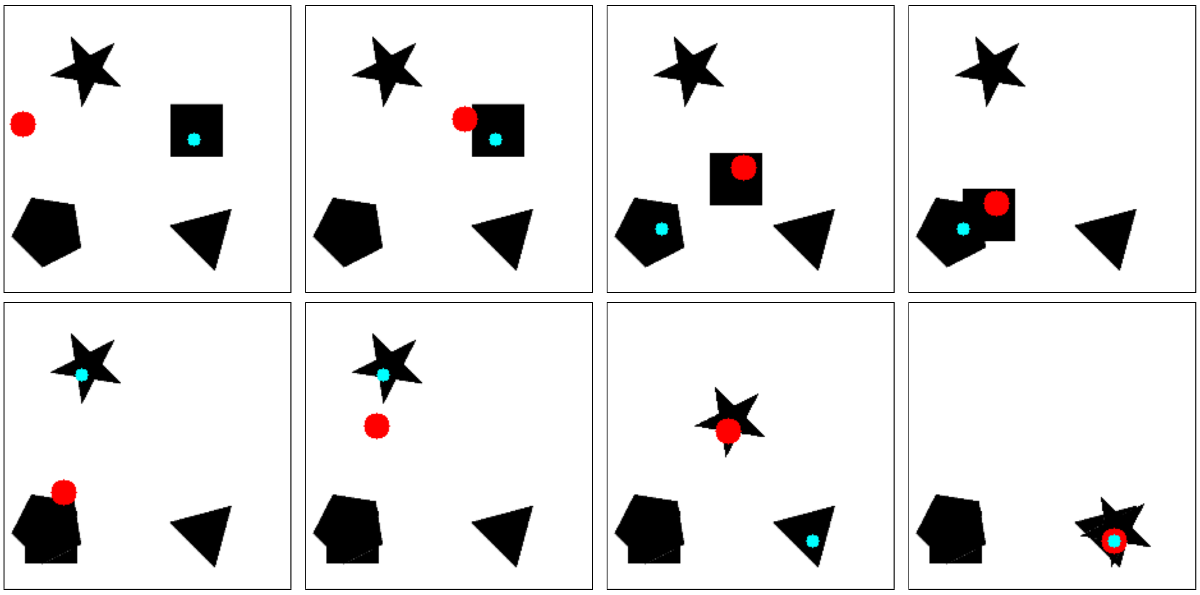


Figure 6.2: The results of SEA Net on the synthetic sequential dataset. The points marked in blue is the prediction of the network. The point in red reflects the end-effector position of the robot. From the top-left corner, the square is predicted as the pick location for the first segment (first and second image in top row). After the square is picked, the pentagon is predicted as the pick/place location (third and fourth image in top row). After this, the star is predicted as the pick location (first and second image in bottom row) followed by the prediction of triangle as the pick/place location (third and fourth image in bottom row).

demonstrations. But the training of these policies takes a long time with a heavy computational resource. Hence pre-existing recorded demonstrations are considered. The pre-existing demonstrations are used to train RL or IL policies. Some of the readily available demonstration could be extracted from Multiple Interactions Made Easy (MIME), Metaworld or Robosuite [46–48]. MIME is a large-scale human demonstration dataset, but it does not contain a variety of sequential tasks [46]. Metaworld consist of a variety of demonstrations performed in a Mujoco physics engine [49], but the demonstrations in Metaworld are manipulation of a single object in the environment. However, Robosuite dataset has many pick-place demonstration. It also has sequential demonstration that could be used in to study the objective [47]. Hence, in this thesis the Robosuite demonstrations are considered to evaluate SEA Net.

Two settings are considered to test the SEA Net. In the first setting, a single object is picked and placed in the required location, as depicted in Fig 6.3. In the second setting, a demonstration with multiple pick and place actions is performed. There are four trajectories in the second setting demonstration.

1. The red square nut is picked
2. The red square nut is placed in its goal location
3. The blue circular nut is picked
4. The blue circular nut is placed in its goal location

This sequence is depicted in Fig 6.4.

To train the SEA Net, a dataset $\mathcal{D} = \{\zeta_1, \zeta_2, ..., \zeta_n\}$ of n expert demonstration is required. Each demonstration $\zeta_i = \{(\mathbf{o}_0, \mathbf{a}_0, \mathbf{c}_0), (\mathbf{o}_1, \mathbf{a}_1, \mathbf{c}_1), ...\}$ where,
$\mathbf{o}_i$ is the visual observations,
$\mathbf{a}_i = \left(\mathbf{T^0}_{pick}, \mathbf{T^1}_{pick}, ... \mathbf{T^k}_{pick}\right)$ for k objects to be manipulate,
$\mathbf{c}_i$ is the label of the object manipulated at the trajectory.

The $\mathbf{a}_i$ consist of pick point ($\mathbf{T}_{pick}$) in each image frames. These pick points are also labels to the SEA Nets which are later converted into one-hot tensor for training the model. To get the label $\mathbf{c}_i$ for

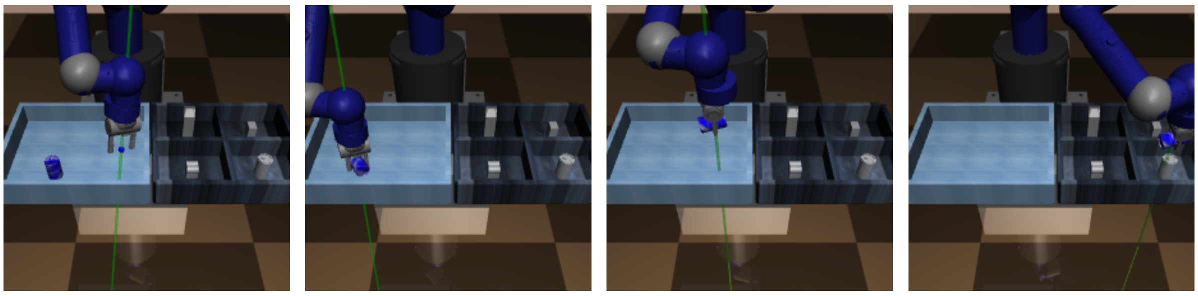


Figure 6.3: The Swayer Robot picks and places a blue colored Can in its corresponding bin. The sequence of the execution are depicted from left to right in the figure.

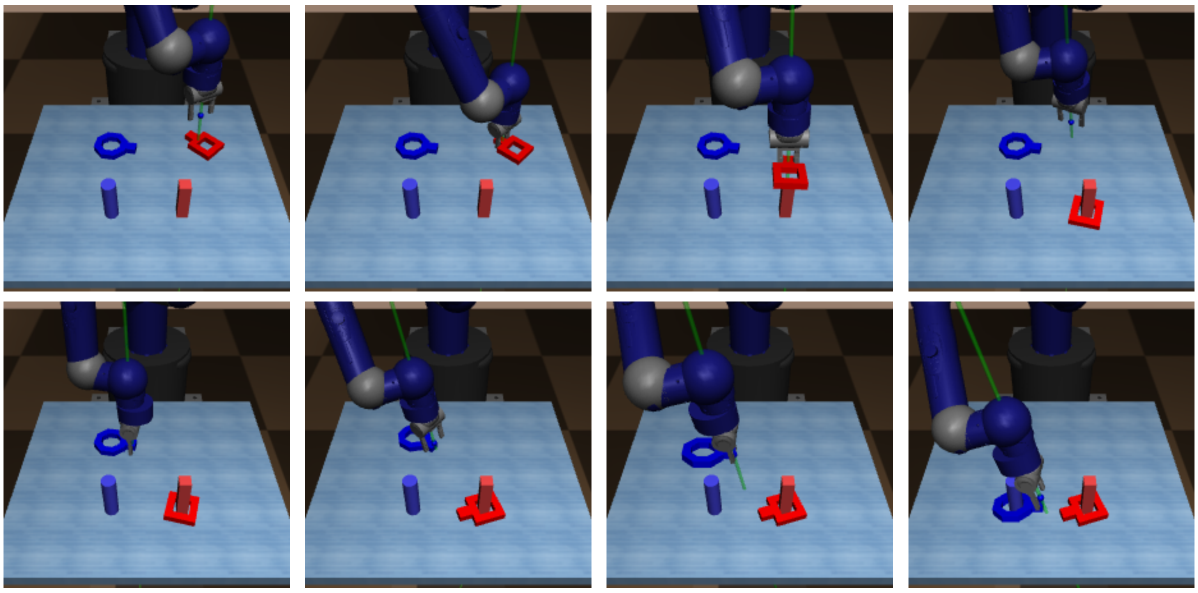


Figure 6.4: The demonstration is carried out by a pre-trained policy. The overall demonstrations could be decomposed into 4 sequences. The first two images in the top row (from left) corresponds to the first sequence. The third and fourth images (from left) on the top row corresponds to the second sequence. The first two images on the bottom row (from left) corresponds to the third sequence. The third and fourth images (from left) in the bottom row corresponds to the fourth sequence

the demonstration, it is necessary to segment the overall demonstration to get individual trajectories. Segmenting the demonstrations means to break a complex movement into a combination of simpler movements, called segments [50]. After segmentation, it is possible to assign a ground-truth value to $\mathbf{c}_i$ that corresponds to the object manipulated in that segment/trajectory. There are various methods to segment a demonstration [51, 52]. In this thesis, the gripper state of the robot was employed to segment trajectories, as it made it possible to obtain required segments in a simple and effective manner. The grippers are only actuated in the robot when the robot arm reaches a particular pick location. The state of the grippers (gs) are recorder throughout the demonstrations. The transition of trajectories occur only when there is a change in the gripper state in time. Therefore, the change in gripper state (S) is calculated between each time steps. The time step corresponding to the change in gripper state are noted. These time steps are then used to label the trajectory which relates with the object being manipulated in that particular trajectory. Algorithm 1 shows the general working of the segmentation algorithm based on the gripper state.

The gripper state may have some noises as depicted in Fig 6.5a. These noises could be due to unintentional actuation during demonstration or re-actuating the gripper to pick the same object again if not appropriately picked. Hence, a simple filter is developed that filters out these noises. This filter take in the gripper state (gs) and removes the noises based on the window size (τ) creating a clear segmentation ($gs_{filtered}$) based on Algorithm 2. This is depicted in Fig 6.5b. Here the sequence 1 is from frame 0 to 1500, sequence 2 is from 1500 to 2200, sequence 3 is from 2200 to 4900 and

Algorithm 1 Segmenting the demonstration using robotic gripper

```
Input: gripper actuation state for whole demonstration ($(gs)$)
Output: Frame number corresponding to change in gripper state $I$
1: for $i$ to $length$(gs) do
2:     if $i$ = 0 then
3:         continue
4:     else
5:         $S \leftarrow (gs_i - gs_{i-1})$          → Compute change in gripper actuation state
6:     end if
7: end for
8: for $i$ to $length$(S) do
9:     if $S(i)$ > 0 then
10:        $I \leftarrow i$          → Save indices where change in state is greater than 0
11:    end if
12: end for
    return $I$
```

sequence 4 is from 4900 till the end.

Algorithm 2 Filtering noises in the gripper state

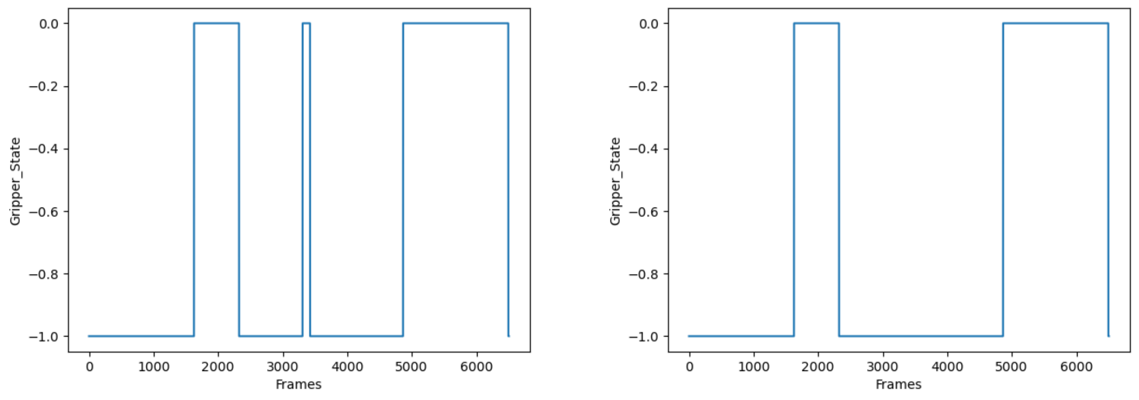
```
Input: gripper actuation state for whole demonstration ($(gs)$), Window size ($(\tau)$), Frame number corresponding to change in gripper state $I$
Output: Filtered Gripper actuation state $gs_{filtered}$
1: for $index$ in $I$ do
2:     C ← 0
3:     for $ws$ in (0 to $\tau$) do
4:         $index \leftarrow index + ws$
5:         if $gs(index)$ = $gs(index + \tau)$ then
6:             C ← C + 1
7:         end if
8:     end for
9:     if C < $\tau$ then
10:        $gs_{filtered}(index + \tau) \leftarrow gs(index - 1)$          → Remove the values with a slight change in gripper states
11:    end if
12: end for
    return $gs_{filtered}$
```

6.3.2. Experimental setup

The SEA Net can learn different spatial locations on manipulation objects with fewer demonstrations. But this is only proven on a synthetic dataset with limited features and a simple end-effector representation in the image. A simulated robot dataset is used for evaluating the model to make the detection more challenging. As mentioned in section 6.3.1, the simulated datasets are extracted from the Robosuite dataset. The images extracted from the simulated robot are challenging because there are a lot of noises and occlusions. These noises and occlusions are due to the movement of the robotic arm in the image frame.

There are two different pick and place tasks used to evaluate the model. The first pick and place task has a single object to be picked and placed, while the second pick and place task has four objects to be manipulated. The first task is shown in Fig 6.3 and the second task is depicted in Fig 6.4.

The overall pipeline for using a simulated robot demonstration is depicted in Fig 6.6. The multiple pick and place task is trained with 10 demonstrations with a total of 2500 images. The single pick and



(a) Unfiltered demonstrations (b) Filtered demonstrations

Figure 6.5: Short time actuation are filtered out.

place task is trained with 10 demonstrations with a total of 1200 images. The completion time for each tasks gives rise to varying number of images.

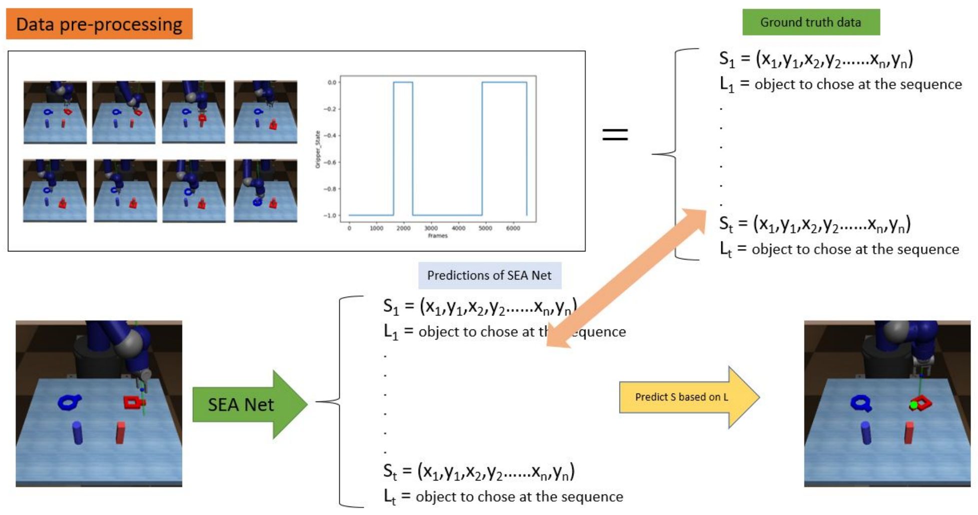

Data pre-processing
Ground truth data
S1 = (x1,y1,x2,y2……xn,yn)
L1 = object to chose at the sequence
St = (x1,y1,x2,y2……xn,yn)
Lt = object to chose at the sequence
Predictions of SEA Net
SEA Net
Predict S based on L

Figure 6.6: The experimental pipeline for simulated robotic dataset is depicted here. The demonstrations are first segmented and the labels are extracted. These labels are then used to train the SEA Net.

6.3.3. Results

Table 6.2 depicts the result of a SEA Net on the two simulated tasks. The single pick-place tasks has a lower accuracy than the multiple pick-place task as it is trained with comparatively lesser observations than the multiple pick-place task. The predictions on the single pick-place task by the SEA Net is depicted in Fig 6.7.

Tasks	Accuracy of the SEA Net in %					
	$\gamma_{threshold}$ = 0.5					
	Classification accuracy	multi-output pick Network(OKS Metric)				Overall Accuracy
		Pick Object 1	Pick Object 2	Pick Object 3	Pick Object 4	
Single pick-place task	65.5	74.4	98.8	-	-	58.8
Multiple pick-place task	88.8	96.5	100	79.5	100	81.8

Table 6.2: The accuracy results of SEA Net with a $\sigma_{threshold}$ of 2 pixels on simulated robot demonstrations for each head

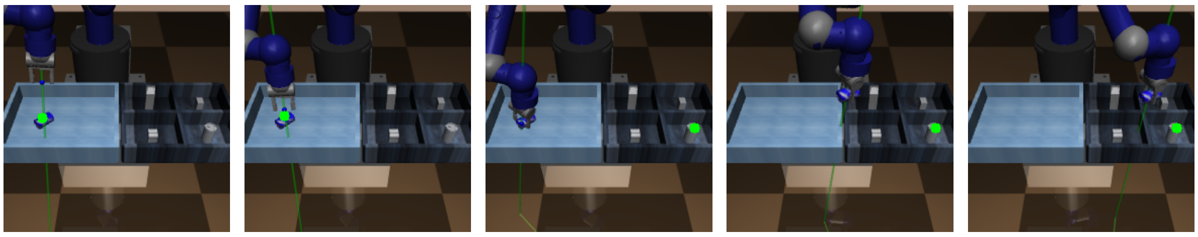


Figure 6.7: The prediction of the SEA Net on the single pick and place task. The green dot represents the prediction of the SEA Net. The sequence is from the left to the right.

The prediction of the SEA Net on multiple pick and place task is significant with a better classification accuracy and overall accuracy. The predictions of the SEA Net in Multiple pick and place task is shown in Fig 6.8.

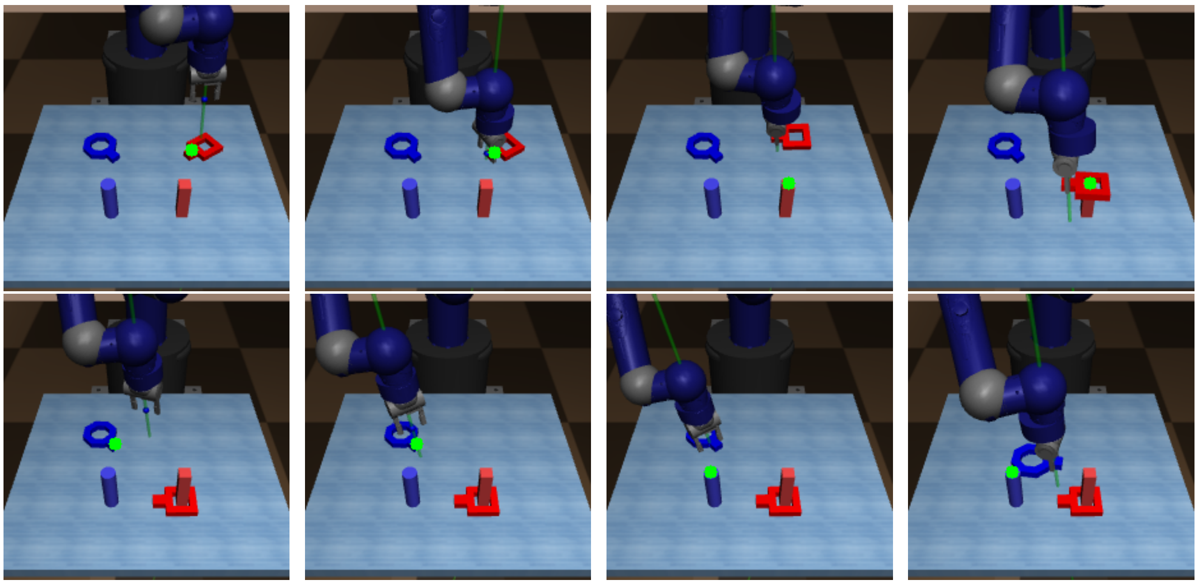


Figure 6.8: The prediction of the SEA Net on the Multiple pick and place task. The green dot represents the prediction of the SEA Net. The sequence starts from the top left image and ends in the bottom right image.

From Table 6.2, the accuracy of the SEA Net for predicting the location of the blue can and blue block in the single and multiple pick-place task, is low. This is due to the presence of an additional blue tracking feature in the end-effector of the robot. The model mispredicts the location of the blue can/block with this additional feature and hence the lower accuracy of the model, as depicted in Fig 6.9.

This misprediction could be overcome by choosing a dataset without any artifacts like the presence of blue feature in the robot arm. Since the dataset used in the thesis are not tailored for the purpose of validating a vision network, these mispredictions are encountered. The SEA Net was also able to predict the pick locations when some of the features are absent in the image. This is depicted in Fig 6.10

In conclusion, the experiments on a simulated robot dataset concludes that the SEA Net is able to generalise well simulated robot dataset. These dataset has a lot of noises. However, the SEA Net was able to predict the pick locations sequentially. This reveals that the model could also be implemented in other simulated robot dataset successfully.

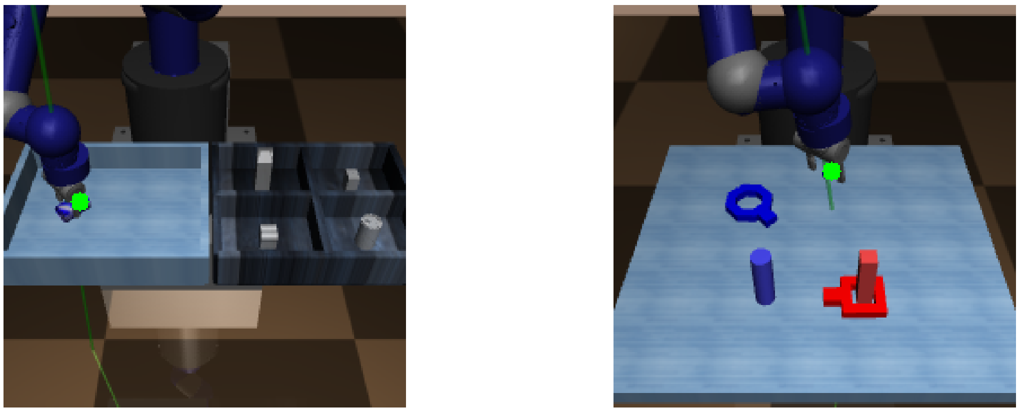

(a) (b)

Figure 6.9: (a) The misprediction in a single pick-place demonstration where the end-effector feature is predicted instead of Can (b) The misprediction in a multiple pick-place demonstration where the end-effector feature is predicted instead of blue block

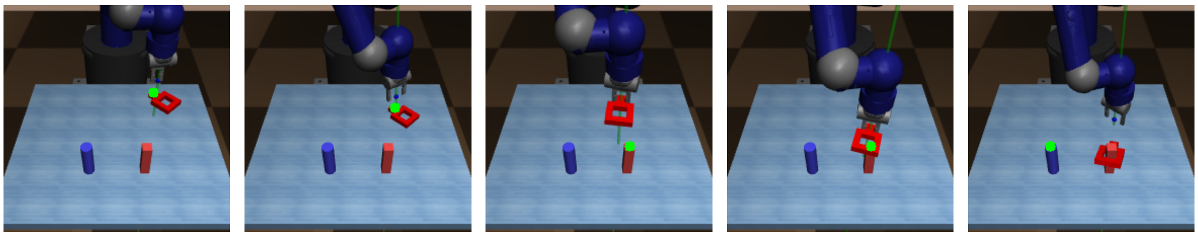


Figure 6.10: The prediction of the SEA Net on the configuration the model was not exposed during training. The sequence starts from the left and ends in the right.

6.4. Experiment - Data-efficiency study

6.4.1. Experimental setup

In this section, the data-efficiency study is conducted on the SEA Net and is compared with other models to evaluate the performance in varying training dataset sizes. Unlike researches that focus on studying the effect of training data on the accuracy of models [53, 54], this study will just analyze the influence of the training data on three models. This comparison study is relevant because the introduction of a fully connected layer in the classification head could decrease the data efficiency as they are not equivariant as a fully convolutional network. Hence, it is essential to test the data efficiency of the SEA Net as the transporter networks are data-efficient. All the comparison models are trained using the same training data sizes, epochs, and learning parameters. The models used are as follows:

- **Regression Model:** A regression model is used to estimate the pick location directly from the images. This straightforward method can be used to predict the spatial locations directly from the configurations. In this model, a ResNet50 backbone is used followed by fully connected layer. The ResNet 43 (used in SEA Net and Transporter Network) has been modified to have a higher receptive field. However, this modification is not required for a regression model. Hence, a ResNet 50 model was chosen because it has a good accuracy with fewer parameters and are readily available [23]. The spatial locations are normalized between 0 to 1, and the loss function used to train this model is a mean square error (MSE) loss function for training the model.

- **Single-stage object detector:**

  A single-stage object detector is explained in detail in Appendix A.2. The Yolo network is a single-stage object detection framework and could detect objects with high effectiveness and efficiency because they are composed of fully convolutional networks [55]. Since the experiment is about testing the data efficiency of the SEA Net, a YOLO framework could be a preferred candidate because of the above-said property. A YOLO model requires the bounding box's center point, width, and height as a ground truth, normalized between 0 to 1. The width and height of the bounding box is assumed to be 25X25 in this experiment as position of the camera is fixed and there is no scaling of objects. The YOLO Model regresses the center point of the bounding box. Therefore, it is necessary to introduce an heuristic in order to get the pick point from YOLO. The simple way of doing this is by choosing the center point of the bounding box as the pick location. This center point used for determining the model's accuracy.

However, a direct out-of-the-box YOLO framework or any object-detection model could not predict the pick points based on the configuration of the environment because the model learns only to detect the presence of a feature in a given image frame.Therefore, the YOLO framework, at a given time instant, will always give the location of all the objects present in the environment through bounding box coordinates. To compare the results with other models, this thesis assumes that the YOLO model can give one pick location at a time instant.

All models are tested on two datasets.

Dual Target dataset

The dual-target dataset used in the previous experiment is again used for a comparative study on data-efficiency of the SEA Net. The performance of the model on Dual Target dataset will show the ability of the model to pick the correct pick location based on the configuration.

Sequential dataset

For the purpose of faster computation time, the sequential dataset is simplified as depicted in Fig 6.11 and is called as the *simplified sequential dataset*. A simplified sequential dataset has two objects present in the object frame. Based on the position of the end-effector (represented in red dot), there will a change in the pick point to be estimated. For Fig 6.11a, the pick point will be the star while for Fig 6.11b, the pick point will be the pentagon. Two different demonstrations are used to train the model to see if the model is able to generalise on two different sequences.

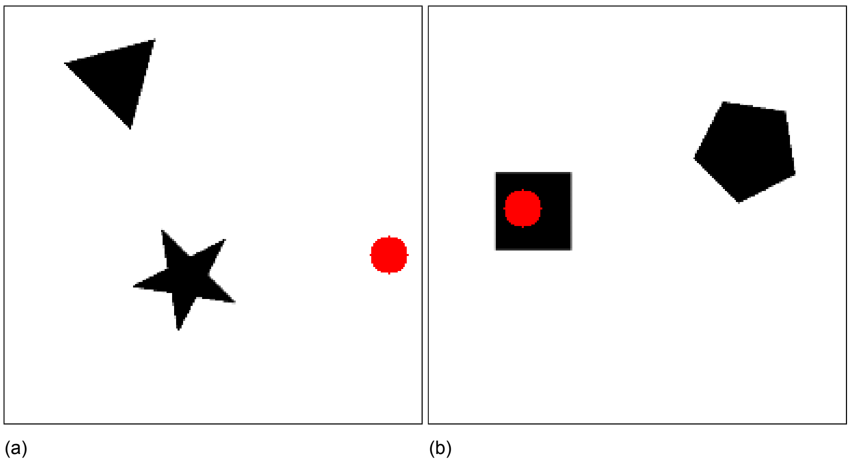

(a) (b)

Figure 6.11: The overall sequence is broken into small sequences. The end-effector is represented by the red dot (a) For this configuration, the goal spatial point is star (b) For this configuration, the goal spatial point is pentagon

Both the dataset has a total of 1000 images which are broken in various stages. The aim of introducing stages is to study the ability of the model to generalise well with lesser sets of data. Therefore, the results will be depicted in five stages as depicted below.

- Stage 1 : 10% of training data

- Stage 2 : 30% of training data
- Stage 3 : 50% of training data
- Stage 4 : 70% of training data
- Stage 5 : 90% of training data

6.4.2. Results

Dual Target Dataset

The accuracy of the models in the data-efficiency analysis on dual target dataset are tabulated in Table 6.3 and 6.4 for a strict and lenient $\sigma_{threshold}$ of 2pixels and 20 pixels, respectively.

Comparison Models	Accuracy of the model in %				
	$\gamma_{threshold}$ = 0.5				
	Stage 1	Stage 2	Stage 3	Stage 4	Stage 5
Regression Model	0	0.2	1.2	1.2	2.6
Modified YOLO Network	14	10.6	18.4	10.8	21.6
SEA Net	**18**	**53.2**	**92.2**	**100**	**100**

Table 6.3: The table depicts the accuracy of the models on a strict $\sigma_{threshold}$ of 2pixels

Comparison Models	Accuracy of the model in %				
	$\gamma_{threshold}$ = 0.5				
	Stage 1	Stage 2	Stage 3	Stage 4	Stage 5
Regression Model	4.8	6.6	39.6	40.6	42.6
Modified YOLO Network*	100	100	100	100	100
SEA Net	**86.4**	**90.8**	**100**	**100**	**100**

Table 6.4: The table depicts the accuracy of the models on a strict $\sigma_{threshold}$ of 20pixels
$*$ Since center of bounding box is assumed as correct prediction in YOLO, the deviation is very less for $\sigma_{threshold}$ of 20 pixels and hence 100% accuracy.

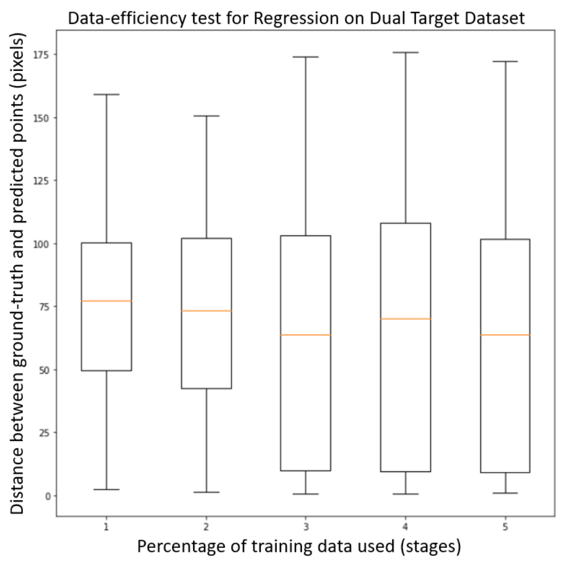
The SEA Net outperforms both the regression and the YOLO Network in all the stages of the data-efficiency test for a $\sigma_{threshold}$ of 2 pixels.

Table 6.5 depicts the performance of the two different heads of a SEA Net on the data-efficiency study. It can be seen that the performance of the classification is head is similar in all the stages of the data-efficiency test. This means that there is no loss in data-efficiency when a classification head is added to the SEA Net. However, the accuracy of the each channel in a multi-output pick network is not similar given each channel has the same amount of data to generalise (refer section 6.2). This depicts that the model generalises differently for different pick locations given they are trained with the same amount of data.

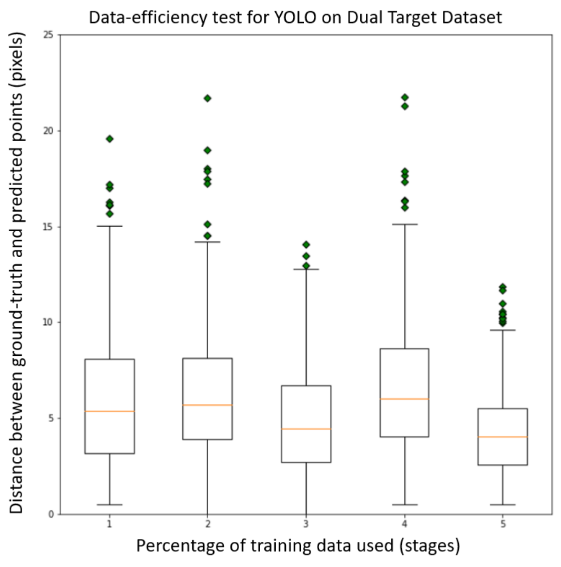
Stages	Accuracy of the SEA Net in %			
	$\gamma_{threshold}$ = 0.5			
	Classification head accuracy	multi-output pick head (OKS Metric)		Overall Accuracy
		Pick Object 1	Pick Object 2	
Stage 1	100	22.2	18.8	18
Stage 2	99.8	60.8	42.8	53.2
Stage 3	100	96.6	85.2	92.2
Stage 4	100	100	100	100
Stage 5	100	100	100	100

Table 6.5: The table depicts the accuracy of each heads in the SEA Net on a strict $\sigma_{threshold}$ of 2pixels

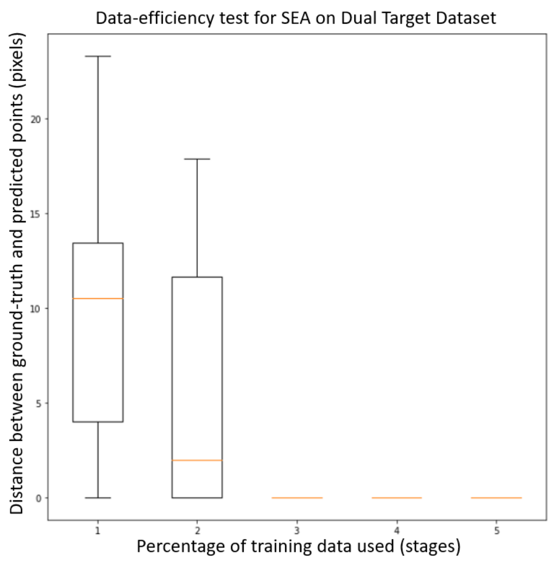
The distribution of the SEA Net's prediction around the required ground truth could be analyzed by the box plots depicted in Fig 6.12 for the dual-target datasets. From Fig 6.12a, it could be seen that the regression model predicts points very far from the ground-truth. However, these points are predicted on the desired pick objects. The distance between the ground-truth and the predicted pick point are not significantly reduced when the model is trained with more data. From Fig 6.12b, it could be seen that the YOLO model outperforms the SEA Net in stage 1. However, it should be noted that the predicted pick point in a YOLO is always assumed to be in the mid-point of the object. Hence, the deviation from the ground-truth is comparatively lesser for YOLO than the SEA Net in stage 1. There is also no significant change in this deviation in various stages of data-efficiency test due to the same reason. From Fig 6.12c, it could be seen that the prediction of the SEA Net is very close to the ground-truth and the model clearly outperforms the other models.



(a)



(b)



(c)

Figure 6.12: The box plots for various stages in the data-efficiency test for dual target dataset. (a) Regression Network. (b) YOLO Network. (c) SEA Net. The box plot depicts the distance of the predicted pick points from the ground-truth on the test images.

Sequential Dataset

The accuracy of the models on sequential dataset are tabulated in Table 6.6 and 6.7 for both strict and lenient $\sigma_{threshold}$ of 2pixels and 20 pixels, respectively.

Comparison Models	Accuracy of the model in %				
	$\gamma_{threshold}$ = 0.5				
	Stage 1	Stage 2	Stage 3	Stage 4	Stage 5
Regression Model	0	1	1	4	6
Modified YOLO Network	14	12	17.4	14.8	22.3
SEA Net	**15**	**26**	**31**	**35**	**70**

Table 6.6: The table depicts the accuracy of the models on sequential dataset on a strict $\sigma_{threshold}$ of 2pixels

Comparison Models	Accuracy of the model in %				
	$\gamma_{threshold}$ = 0.5				
	Stage 1	Stage 2	Stage 3	Stage 4	Stage 5
Regression Model	7	40	49	54	78
Modified YOLO Network	100	100	100	100	100
SEA Net	**59**	**62**	**71**	**80**	**87**

Table 6.7: The table depicts the accuracy of the models on sequential dataset on a strict $\sigma_{threshold}$ of 20pixels.
$*$ Since center of bounding box is assumed as correct prediction in YOLO, the deviation is very less for $\sigma_{threshold}$ of 20 pixels and hence 100% accuracy.

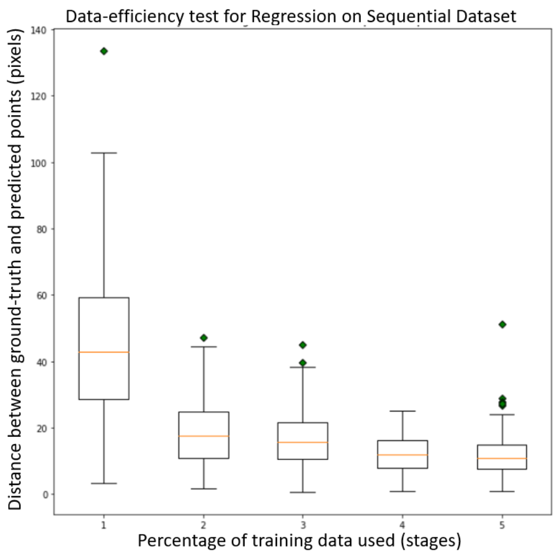
The SEA Net performs well on a sequential dataset when compared to the other two models. The same trend as discussed in the previous section is observed from the Table. However, the performance of the SEA Net on the sequential dataset is lower when compared to the dual-target dataset. This trend could be because the SEA Nets are trained with two different demonstrations, as shown in Fig 6.11. This means that each channel in a SEA Net needs to generalize for two different shapes (i.e, one channel needs to generalise on square and star while the other channel needs to generalise on pentagon and triangle), unlike the dual-target dataset where each channel generalises on one shape. However, it could be seen that the SEA Net is still able to predict the sequences with higher accuracy when trained with different demonstrations.

Stages of Data-efficiency Test	Accuracy of the SEA Net in %		
	$\gamma_{threshold}$ = 0.5		
	Classification accuracy	Multi-Output Pick Network(OKS Metric)	
		Pick Object 1	Pick Object 2
Stage 1	98	55	14
Stage 2	98	63	20
Stage 3	99	61	38
Stage 4	100	58	57
Stage 5	100	70	97

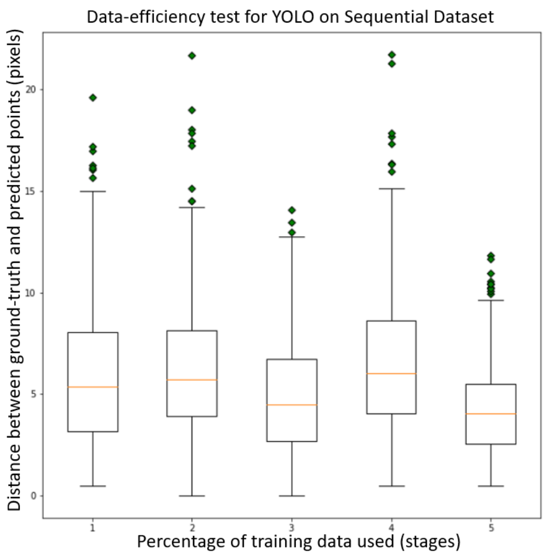
Table 6.8: The table depicts the accuracy of each head in the SEA Net on the various stages of data-efficiency study.

The SEA Net classification head again performs well for all the stages of the data-efficiency study as depicted in Table 6.8. However, the accuracy of the output channel of the multi-output pick head increases with an increase in the amount of data. The distribution of the SEA Net's prediction around the required ground truth could be analyzed by the box plots depicted in Fig 6.13 for the sequential datasets.

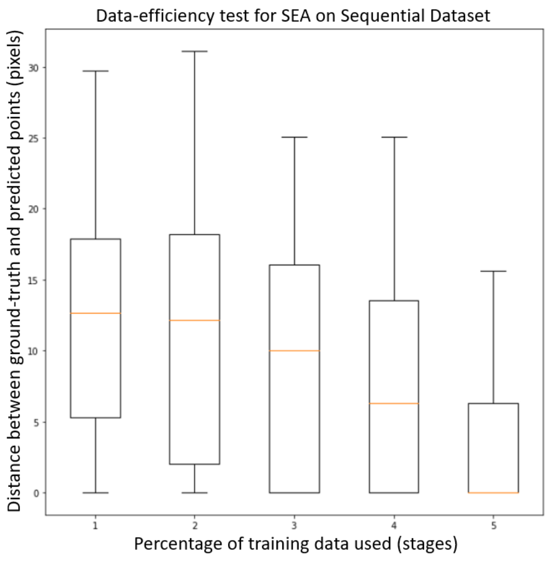
For simplified sequential datasets as depicted in Fig 6.13a, the regression model does not show promising results. The observation is similar to that of the dual-target dataset. The range of deviation from the ground truth predictions is very high compared to the YOLO and the SEA Net. For a strict threshold of 2-pixel deviation, the regression model has a very low accuracy but for a lenient threshold,



(a)



(b)

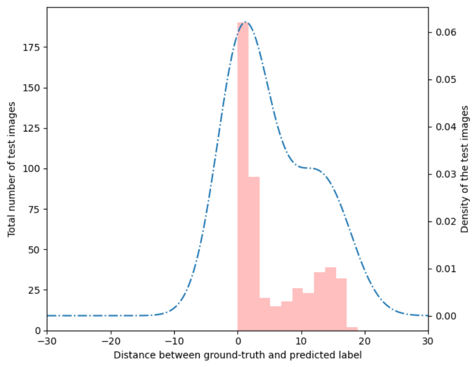


(c)

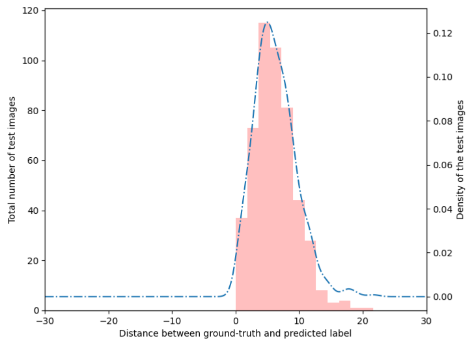
Figure 6.13: The box plots for various stages in the data-efficiency test for simplified sequential dataset. (a) Regression Network. (b) YOLO Network. (c) SEA Net. The box plot depicts the distance of the predicted pick points from the ground-truth on the test images

the model can have good accuracy, which means that the model can still predict the pick locations on the required object in the environment. The regression model might perform considerably better when provided with more training data or increasing the number of iterations, but this is out of the scope of the thesis.

The range of deviations between ground truth and prediction for YOLO (in Fig 6.13b) and SEA Net (Fig 6.13c) is much lower (i.e., in the range of 0 to 20 pixels). However, the accuracy of the SEA Net is more than YOLO in all the stages of the data-efficiency test. Looking closer into these results, the SEA Net can predict more samples closer to a deviation of 0 than the YOLO model. This is depicted in Fig 6.14, where the number of samples closer to zero deviation is more in the SEA Net than the YOLO model. Hence, the SEA Net's generalization is far better than both the models in the lower stages of the data-efficiency test, where the number of training samples used to train the networks are significantly less. The SEA Net's prediction is highly accurate in higher stages of the data-efficiency test, where the model completely outperforms the other two models.



(a)



(b)

Figure 6.14: (a) The distribution curve for SEA Net on 30% training data. (b) The distribution curve for YOLO Net on 30% training data. The distribution curve depicts clearly that the spatial locations predicted by SEA Net are closer to the ground truth than predictions from YOLO.

In conclusion, from the data-efficiency test results it is evident that the SEA Net is able to generalise well with a fewer demonstrations when compared to the other models. However, some drawback of the SEA Net model could also be noted. The performance of the SEA Net dropped when a larger sequence with more trajectories are considered for prediction. But the performance of the SEA Net is still better than the other models for a larger sequence demonstration. In the next section the SEA Net is studied on dataset from the simulated robot.

6.5. Discussion

The SEquential Attention Network (SEA Net) was introduced to predict the pick locations sequentially without depending on the receptive field of the network. Previously, the Transporter Networks were used to predict the pick location sequentially. Experiments on the Transporter Network revealed that the predicted outputs were multi-modal and depends on the receptive field of the network to solve sequential tasks. The SEA Nets were able to overcome the aforementioned problem by introducing a multi-output pick head and classification head. To perform experiments on the SEA Net, datasets were created/designed in this thesis. A simple synthetic dataset were created using simple shapes and a simulated robot dataset were created using pre-existing demonstrations. The pre-existing demonstrations were broken into segments using a simple segmentation algorithm.

Experiments on dual-target dataset revealed that SEA Nets chose the desired pick location based on the configuration of the environment with a good accuracy. The experiments on the sequential dataset depict that the SEA Net was able to consecutively predict the pick locations based on the change in the configuration in the environment. However, the accuracy of the overall SEA Net was dependent on the classification head i.e., the overall accuracy of SEA Net will reduce when the accuracy of the classification head is low. It was also observed that SEA Net's performance decreased with an increase in the number of objects to be picked in a sequence. However, this accuracy could be increased when provided with more data. Experiments on the simulated dataset showed that the SEA Net model could perform well on noisy and complex datasets. The model was able to predict the pick location even if some features in the environment were absent. However, there were mispredictions due to artifacts present in the dataset. This limitation could be overcome by choosing a tailored-made dataset from simulators or gazebos.

The data-efficiency study shows that the introduction of the classification head did not affect the performance of the SEA Net using fewer demonstration. SEA Net performs better than the regression and object detection model with fewer data. The SEA Net predictions were close to the required ground truth, whereas the object detection model always assumed the predictions as the mid-point of the bounding box.

7 Conclusion and Future Works

This thesis aims to extract high-level goals from images so that a low-level controller can complete the overall task. The aim was to learn these high-level goals from images given by human demonstrations.

This thesis proposes a novel model called a SEquential Attention Network (SEA Net) that can sequentially predict the pick locations for completing a sequential task with fewer demonstrations. Initially, the pick network of the Transporter Network was tested. These network has a larger receptive field and the expected output from these models should be dependent on every pixel in the input image. However, from preliminary experiments the pick network predicted multi-modal outputs, which means that the outputs were predicted based on the neighbouring pixels rather than the whole configuration. This revealed that the pick network of the Transporter network was dependent on the receptice field size to solve sequential tasks.

To get uni-modal outputs for the same receptive field size, the multi-output pick network was introduced in this thesis. The architecture of the multi-output pick network separated N-modal output to N unimodal outputs. However, the multi-output pick network could not still learn to give a single uni-modal solutions for a given image. Therefore, a classification head was added to the multi-output pick network to introduce the SEA Nets. The SEA Nets were able to give a single uni-modal solution for the given image. The dataset were created/designed to evaluate the SEA Net. The synthetic dataset were to depict sequential tasks using simple shapes, while pre-existing demonstrations were used to generate simulated robot dataset. A simple segmentation algorithm was developed to label the ground truth for the classification head. The performance of the model was evaluated using OKS Metrics.

The SEA Net was able to give uni-modal solutions for given time step based on the configuration of the objects in the image. SEA Nets were data-efficient when compared to regression and YOLO networks. However, the networks accuracy depends on the classification head. If the classification accuracy is low, the overall accuracy of the network tends to be low. The model also assumes that the number of objects manipulated in the sequential tasks is known beforehand. But, SEA Nets are independent of the size of receptive field for solving sequential tasks.

7.1. Limitations and Future Work

This section presents the limitation of the thesis.

- The SEA Nets are not validated on the real world environment on real robot. There could be many challenges in implementing the algorithm on real world robot like choosing a correct segmentation algorithm, creating a ROS communication. But these aspects were not explored using SEA Nets and could be looked into in future.
- The SEA Net assumes that it has prior knowledge about the number of objects that is being manipulated in the given demonstrations. This assumption could make the model less flexible for online training.
- The SEA Net could struggle when the sequence to predict is large or there are a large number of pick points. It could be an interesting path to explore ways where this limitation could be overcomed.
- In this thesis, the pick points for each frames were labelled because the camera parameters of the Mujoco simulators were not accurate to convert the end-effector position of the robot to the pixel location in the camera. This work was tedious as human annotation were required. With the correct camera parameters, a self supervised method could be devised that uses a segmentation algorithm to also extract the position of the objects in the image. This aspect could also be looked in for future works.
- An appropriate motion primitive was not used in this thesis. In future, it could be interesting to explore the possibility of using more than one motion primitive and choosing the appropriate motion primitive from the output of the classification head of the SEA Net.

A Appendix

A.1. Data-Driven Control

A.1.1. Markov Decision Process

A *Markov decision process (MDP)* is a discrete-time, finite learning framework (can also be ”continuous” which is ignored in this thesis) in which the current state does not depend on the history of the previous states. Consider the agent in Fig A.1. In a discrete-time MDP framework, an agent interacts with an environment at discrete time scale $t = 0, 1, 2, \ldots$ In each of these time steps, the agent observes the environment and extracts the state of the environment $s_t \in \mathcal{S}$. Based on the state, the agent executes an action $a_t \in \mathcal{A}$ which results in a reward r_{t+1} being awarded in the next time-step and the transition of the state from s_t to s_{t+1} [5].

The transition probability is given by

$$P_a(s, s') = \Pr(s_{t+1} = s' \mid s_t = s, a_t = a) \tag{A.1}$$

The Markov decision process is hence a 4-tuple (S, A, P_a, R_a). To better understand the MDP, consider a tomato harvester used for picking ripe tomatoes in a greenhouse. The tomato harvester has a set of states and actions. For each time step, the tomato harvester observes the environment, perceives the tomatoes’ state, and selects an action based on the policy. The MDP is the basic for reinforcement learning and imitation learning algorithm.

In **reinforcement learning**, the agent learns the policy by maximizing the cumulative reward function. [56–58]. One of the disadvantage of reinforcement learning is designing and implementing a good reward function. A feasible and practical solution to overcome this problem is *Imitation Learning*.

In **imitation learning**, the agent has access to the expert’s demonstrations, i.e., the state and the action of the expert agent.[44] In short, the agent has access to the expert’s policy. Some imitation learning also has the expert available during training, known as interactive learning. The agent learns to imitate the expert trajectories via various principles. Some of the famous principles used in imitation learning are: 1. Behavioral cloning [4, 59–61]: learning the expert’s policy by minimising the error between the agent’s policy and the expert’s policy 2. Inverse reinforcement learning [41, 60, 62]: learning

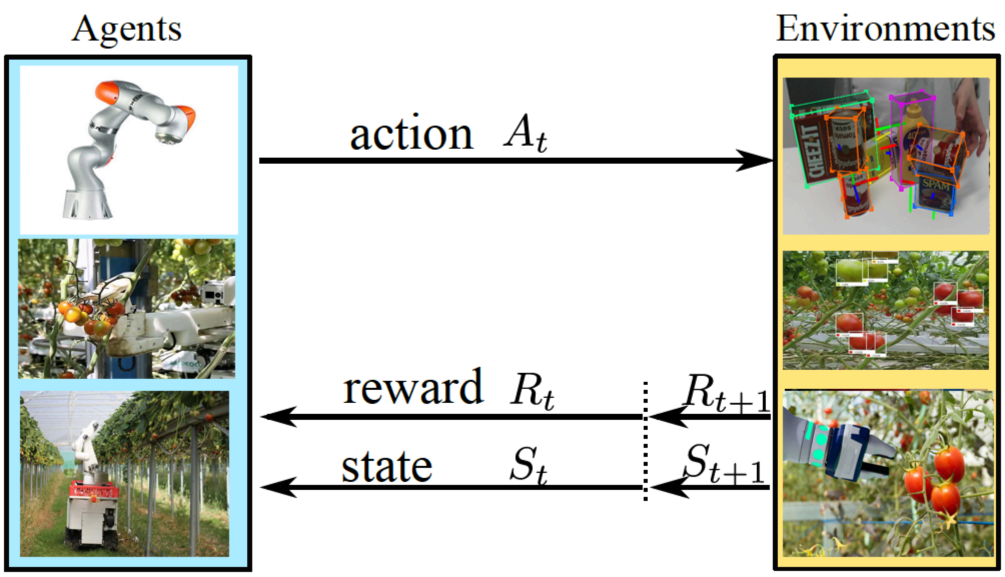


Figure A.1: The agent learning from the environment based on MDP

a reward function based on the expert's trajectories.

A.1.2. Hierarchical Imitation Learning

To understand hierarchical learning, consider an example of a robot agent whose objective is to move from the Delft University campus to the Delft station. As mentioned in the previous section, the agent needs to consider a large search space to achieve the goal successfully. In a hierarchical algorithm, the overall goal is divided into sub-goals and, in achieving these sub-goals attain temporal abstraction.[5] In the example, instead of directly reaching the Delft station, the robot could have intermediate goals, as depicted in Fig A.2. By decomposing the overall goals into sub-goals, the large search space will reduce significantly.

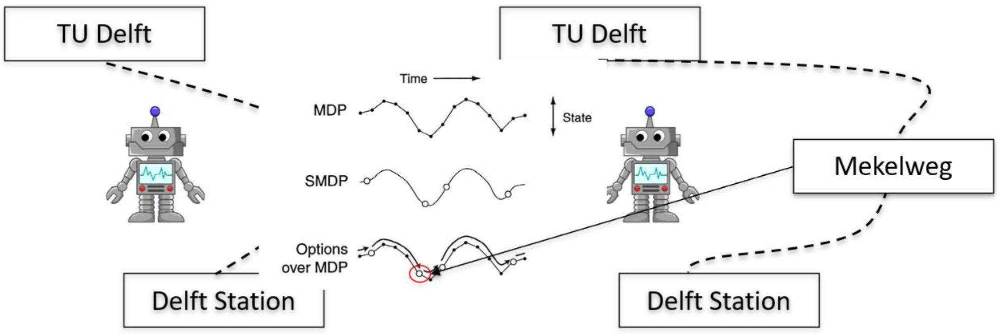


Figure A.2: The agent moving from TU Deflt to Delft station has sub-goals (Mekelweg) in a hierarchical control

Hierarchical learning is based on the Semi-Markov decision process (SMDP). In SMDP, *options* are a set of low-level actions performed based on MDPs at discrete-time steps.[5] The options could also be called sub-controllers. Options usually consists of three components namely, a policy that maps state to action $\pi : \mathcal{S} \times \mathcal{A}$, a termination condition $\beta : \mathcal{S}^{+} \rightarrow [0, 1]$ and an initiation set $\mathcal{I} \subseteq \mathcal{S}$ [5]. The option starts when the state observed is present in the initiation set $s_t \subseteq \mathcal{I}$. After the initiation of the option, the policy associated with the option is executed. The option is then terminated when it meets the termination condition in a policy. Hence in hierarchical learning, the high-level controller aims to extract high-level states as depicted in the central image of Fig A.3. Based on the extracted states, the possible low-level controller(i.e., options) is picked (based on the state present in the initiation set) to carry out a particular task in a sequence.

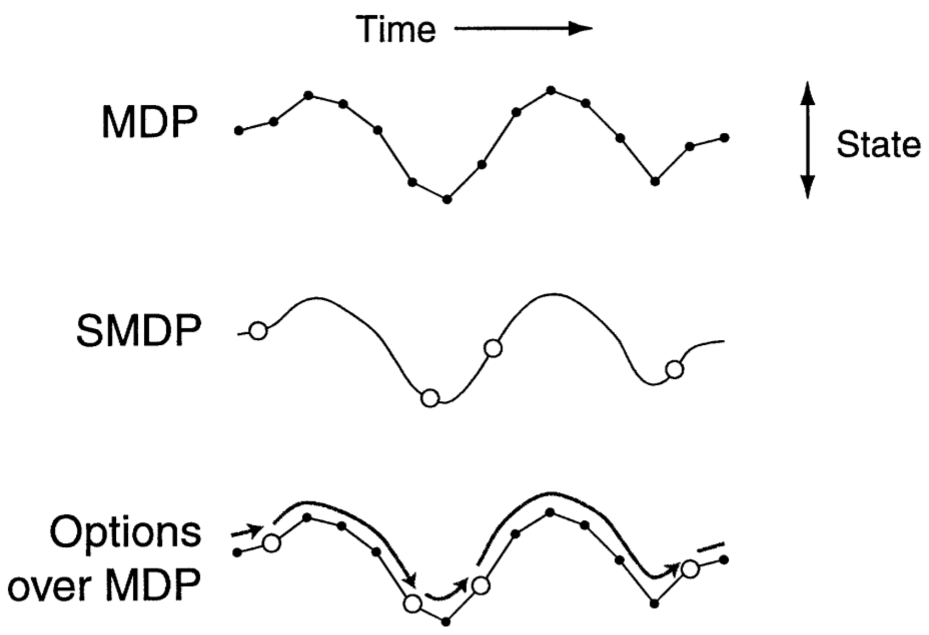


Figure A.3: The state trajectory for MDPs, SMDPs and the effect of options in the SMDPs [5]

Hierarchical learning is implemented using both reinforcement learning and imitation learning. The hierarchical framework that uses a reinforcement learning algorithm to learn high-level and low-level controllers is Hierarchical Reinforcement Learning (HRL). To know more about HRL and its various types refer [63–67] .

In contrast, in hierarchical imitation learning, the low-level controller is learned using an imitation learning algorithm. The high-level controller of a HIL algorithm could be learned using a reinforcement learning algorithm or imitation learning algorithm.[68]

If, in addition to trajectories, the expert demonstrator provides observation as images, a high-level controller could translate the images to the robot's state with the help of a computer vision technique. This technique is used in algorithms that learn a sequence of actions by using context learning. Context learning is a type of HIL where the actions are abstracted into symbols or texts. [9, 10]. Each of these symbols represents a policy that the agent needs to execute. The high-level controller here will implement a computer vision algorithm to estimate the position or the pose of the object the robot needs to manipulate. This position or pose are translated to a target state for the robot to achieve. In [9], a pre-trained Mask RCNN is used as the high-level controller, which gives the target pose of the object or implicitly the target pose of the robot arm. Likewise, in [10] a SegNet is used for extracting the target pose of the object that is to be manipulated. This research uses a pre-trained existing computer vision model as the high-level controller that gives target states to the robot arm. Generative networks [69] are used to generate sub-goal images given the final goal image to the model. [8] These sub-goals are generated in such a way that total cost for the robot to move through these sub-goals is less for completing the sequence. It is also possible to solve long-horizon tasks with planning and sequencing algorithms.[70]

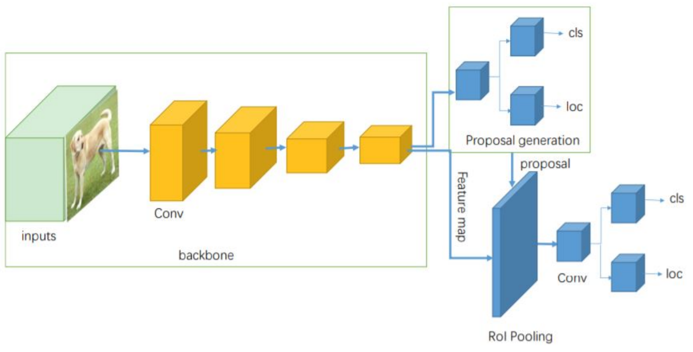
In conclusion, an hierarchical learning needs two-controllers namely a high-level and low-level controller. The high-level controller will give the goal states that the robot needs to transverse. The spatial location that would be extracted using computer-vision techniques could be used as the goal state of the high-level controller. Hence, a low-level controller is assumed to take the spatial locations in image co-ordinate and execute a policy to move the robot to the corresponding location in the world co-ordinate.

A.2. Object detection and segmentation models

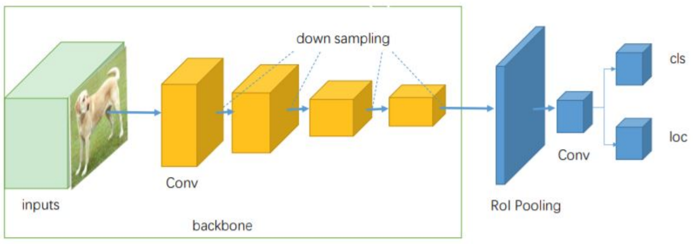
Object Detection

Traditional object detection frameworks are done using handcrafted features, but it was addressed with deep learning. Hence, the problem definition of an object detection framework is to estimate the location of the objects in the images and classify the object's category. The object detection algorithms could be classified into two types:

1. **Two-stage Detectors:** A category-independent region of proposal (ROP) is extracted from the image in the first stage in a two-stage detector. The second stage is to classify and regress the bounding box of the region of proposals.[6] The most popular two-stage detectors are RCNN [71], Fast-RCNN, Faster RCNN [72], and Mask RCNN[73]. In addition to object detection, a Mask RCNN is used to generate binary masks of the feature in the region of proposals. A Mask-RCNN is used in instance segmentation of the objects.
2. **Single-stage Detectors:** Unlike a two-stage detector, the image is completely regressed to the bounding box position and classification in a single-stage detectors. Single-stage detectors could be compared to an end-to-end approach with no intermediate network for predicting an intermediate result. The most popular single-stage detector is YOLO and SSD [7, 74].



(a) A representation of a two-stage object detector.



(b) A representation of a single-stage object detector.

Figure A.4: A two-stage detector has a separate region of proposal network while a single-stage detector directly regresses the image [6]

In robotic manipulation, these two-stage detectors are used to estimate the object's position in the scene. A two-stage detector is used in manipulation techniques because it is more accurate. But the amount of training data required to train these networks are heavy. These methods are light and could be trained with lesser training data when compared to a two-stage network. But the accuracy of a single-stage detector is comparatively lesser than a two-stage network.

Segmentation

Image segmentation is used to segment the images into desired regions or objects. Segmentation tasks are generally used in medical segmentation, autonomous vehicles etc. Segmentation involves the classification of each pixel in an image to a particular class. It is also used to partitioning individual objects(instance segmentation). Segmentation has been used in many applications. The backbone of segmentation can be either a CNN for computer vision or Recurrent neural networks(RNN). But using an RNN for segmentation task is generally not used given the speed of processing the image. An RNN is used for sequential data learning, and since this cant be parallelized for computation, it is difficult to achieve the same speed a CNN achieves [75]. Segmentation could be classified as semantic segmentation and instance segmentation.

Semantic Segmentation: Semantic segmentation are used to predict the object boundaries by predicting the class label for each pixel in the image. The final prediction of a semantic segmentation model is the prediction of meaningful clusters of prediction. The semantic segmentation model could predict the class label for each pixel by using skip connections. These skip connections are used to preserve the spatial information while predicting the segmentation maps. UNet [76] is a popular segmentation technique that uses the skip connection for predicting the class of each pixels. It is also possible to predict segmentation maps by using a encoder-decoder structure. The main different between a skip based and decoder based structure is that, in a decoder based structure, the pooling indices in encoder stage are learned in the decoder stage. The SegNet [77] architecture uses a encoder-decoder structure for generating segmentation maps.

The semantic segmentation models however doesnot predict the difference between the instances of the objects present in the image. Hence the instance segmentation model was introduced.

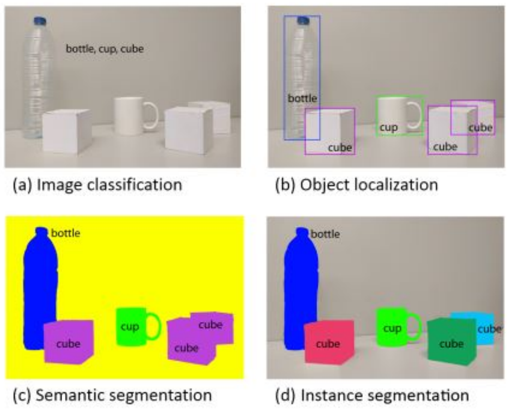


Figure A.5: The difference between semantic segmentation and instance segmentation is depicted in (c) and (d) [78]

Instance Segmentation: Instance segmentation method is the next step of a semantic segmentation method. It is used to predict the different instances of the same class in an image. However, this could be challenging as the number of instance of the same class are unknown. Instance segmentation provides extra information which could be used to know about occlusions or detecting a particular object of the same class for robotic grasping.

Bibliography

[1] Amir H Abdi, Pramit Saha, Venkata Praneeth Srungarapu, and Sidney Fels. Muscle excitation estimation in biomechanical simulation using naf reinforcement learning. In *Computational Biomechanics for Medicine*, pages 133–141. Springer, 2020.

[2] JongYoon Lim, Ho Seok Ahn, Mahla Nejati, Jamie Bell, Henry Williams, and Bruce A MacDonald. Deep neural network based real-time kiwi fruit flower detection in an orchard environment. *arXiv preprint arXiv:2006.04343*, 2020.

[3] Javad Ghofrani, Robert Kirschne, Daniel Rossburg, Dirk Reichelt, and Tom Dimter. Machine vision in the context of robotics: A systematic literature review. *arXiv preprint arXiv:1905.03708*, 2019.

[4] Brenna D Argall, Sonia Chernova, Manuela Veloso, and Brett Browning. A survey of robot learning from demonstration. *Robotics and autonomous systems*, 57(5):469–483, 2009.

[5] Richard S Sutton, Doina Precup, and Satinder Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1-2):181–211, 1999.

[6] Licheng Jiao, Fan Zhang, Fang Liu, Shuyuan Yang, Lingling Li, Zhixi Feng, and Rong Qu. A survey of deep learning-based object detection. *IEEE access*, 7:128837–128868, 2019.

[7] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767*, 2018.

[8] Suraj Nair and Chelsea Finn. Hierarchical foresight: Self-supervised learning of long-horizon tasks via visual subgoal generation. *arXiv preprint arXiv:1909.05829*, 2019.

[9] Sören Pirk, Karol Hausman, Alexander Toshev, and Mohi Khansari. Modeling long-horizon tasks as sequential interaction landscapes. 6 2020.

[10] Shuo Yang, Wei Zhang, Weizhi Lu, Hesheng Wang, and Yibin Li. Learning actions from human demonstration video for robotic manipulation. 9 2019.

[11] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[12] Chigozie Nwankpa, Winifred Ijomah, Anthony Gachagan, and Stephen Marshall. Activation functions: Comparison of trends in practice and research for deep learning. *arXiv preprint arXiv:1811.03378*, 2018.

[13] B.L. Kalman and S.C. Kwasny. Why tanh: choosing a sigmoidal function. In *[Proceedings 1992] IJCNN International Joint Conference on Neural Networks*, volume 4, pages 578–581 vol.4, 1992.

[14] Dabal Pedamonti. Comparison of non-linear activation functions for deep neural networks on mnist classification task. *arXiv preprint arXiv:1804.02763*, 2018.

[15] Saad Albawi, Tareq Abed Mohammed, and Saad Al-Zawi. Understanding of a convolutional neural network. In *2017 International Conference on Engineering and Technology (ICET)*, pages 1–6. Ieee, 2017.

[16] Hung Le and Ali Borji. What are the receptive, effective receptive, and projective fields of neurons in convolutional neural networks? *arXiv preprint arXiv:1705.07049*, 2017.

[17] Wenjie Luo, Yujia Li, Raquel Urtasun, and Richard Zemel. Understanding the effective receptive field in deep convolutional neural networks. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, pages 4905–4913, 2016.

[18] Songtao Liu, Di Huang, et al. Receptive field block net for accurate and fast object detection. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 385–400, 2018.

[19] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *IEEE transactions on pattern analysis and machine intelligence*, 40(4):834–848, 2017.

[20] Alejandro Newell, Kaiyu Yang, and Jia Deng. Stacked hourglass networks for human pose estimation. In *European conference on computer vision*, pages 483–499. Springer, 2016.

[21] Bin Xiao, Haiping Wu, and Yichen Wei. Simple baselines for human pose estimation and tracking. In *Proceedings of the European conference on computer vision (ECCV)*, pages 466–481, 2018.

[22] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[23] Alfredo Canziani, Adam Paszke, and Eugenio Culurciello. An analysis of deep neural network models for practical applications. *arXiv preprint arXiv:1605.07678*, 2016.

[24] Weibo Liu, Zidong Wang, Xiaohui Liu, Nianyin Zeng, Yurong Liu, and Fuad E Alsaadi. A survey of deep neural network architectures and their applications. *Neurocomputing*, 234:11–26, 2017.

[25] Vincent Lepetit, Francesc Moreno-Noguer, and Pascal Fua. Epnp: An accurate o (n) solution to the pnp problem. *International journal of computer vision*, 81(2):155, 2009.

[26] Stefan Hinterstoisser, Vincent Lepetit, Slobodan Ilic, Stefan Holzer, Gary Bradski, Kurt Konolige, and Nassir Navab. Model based training, detection and pose estimation of texture-less 3d objects in heavily cluttered scenes. In *Asian conference on computer vision*, pages 548–562. Springer, 2012.

[27] Antti Hietanen, Jyrki Latokartano, Alessandro Foi, Roel Pieters, Ville Kyrki, Minna Lanz, and Joni-Kristian Kämäräinen. Object pose estimation in robotics revisited. *arXiv preprint arXiv:1906.02783*, 2019.

[28] Yao Wang, Ying Xu, Xiaohui Zhang, Zhen Sun, Yafang Zhang, Guoli Song, and Junchen Wang. 3d pose estimation for robotic grasping using deep convolution neural network. In *2018 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, pages 513–517. IEEE, 2018.

[29] Meng Tian, Liang Pan, Marcelo H Ang, and Gim Hee Lee. Robust 6d object pose estimation by learning rgb-d features. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6218–6224. IEEE, 2020.

[30] Xinke Deng, Yu Xiang, Arsalan Mousavian, Clemens Eppner, Timothy Bretl, and Dieter Fox. Self-supervised 6d object pose estimation for robot manipulation. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3665–3671. IEEE, 2020.

[31] He Wang, Srinath Sridhar, Jingwei Huang, Julien Valentin, Shuran Song, and Leonidas J Guibas. Normalized object coordinate space for category-level 6d object pose and size estimation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2642–2651, 2019.

[32] Peiyuan Ni, Wenguang Zhang, Xiaoxiao Zhu, and Qixin Cao. Pointnet++ grasping: Learning an end-to-end spatial grasp generation algorithm from sparse point clouds. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3619–3625. IEEE, 2020.

[33] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 652–660, 2017.

[34] Xingyu Liu, Rico Jonschkowski, Anelia Angelova, and Kurt Konolige. Keypose: Multi-view 3d labeling and keypoint estimation for transparent objects. 12 2019.

[35] Lucas Manuelli, Wei Gao, Peter Florence, and Russ Tedrake. kpam: Keypoint affordances for category-level robotic manipulation. 3 2019.

[36] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014.

[37] Chelsea Finn, Xin Yu Tan, Yan Duan, Trevor Darrell, Sergey Levine, and Pieter Abbeel. Deep spatial autoencoders for visuomotor learning. 9 2015.

[38] Kevin Zakka, Andy Zeng, Johnny Lee, and Shuran Song. Form2fit: Learning shape priors for generalizable assembly from disassembly. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 9404–9410. IEEE, 2020.

[39] Andy Zeng, Pete Florence, Jonathan Tompson, Stefan Welker, Jonathan Chien, Maria Attarian, Travis Armstrong, Ivan Krasin, Dan Duong, Vikas Sindhwani, et al. Transporter networks: Rearranging the visual world for robotic manipulation. *arXiv preprint arXiv:2010.14406*, 2020.

[40] Lars Berscheid, Pascal Meißner, and Torsten Kröger. Self-supervised learning for precise pick-and-place without object model. *IEEE Robotics and Automation Letters*, 5(3):4828–4835, 2020.

[41] Timothée Lesort, Natalia Díaz-Rodríguez, Jean-Franois Goudou, and David Filliat. State representation learning for control: An overview. *Neural Networks*, 108:379–392, 2018.

[42] Matteo Ruggero Ronchi and Pietro Perona. Benchmarking and error diagnosis in multi-instance pose estimation. In *Proceedings of the IEEE international conference on computer vision*, pages 369–378, 2017.

[43] Kevin P Murphy. *Machine learning: a probabilistic perspective*. MIT press, 2012.

[44] Takayuki Osa, Joni Pajarinen, Gerhard Neumann, J Andrew Bagnell, Pieter Abbeel, and Jan Peters. An algorithmic perspective on imitation learning. *arXiv preprint arXiv:1811.06711*, 2018.

[45] The garage contributors. Garage: A toolkit for reproducible reinforcement learning research. `https://github.com/rlworkgroup/garage`, 2019.

[46] Pratyusha Sharma, Lekha Mohan, Lerrel Pinto, and Abhinav Gupta. Multiple interactions made easy (mime): Large scale demonstrations data for imitation. In *Conference on robot learning*, pages 906–915. PMLR, 2018.

[47] Yuke Zhu, Josiah Wong, Ajay Mandlekar, and Roberto Martín-Martín. robosuite: A modular simulation framework and benchmark for robot learning. *arXiv preprint arXiv:2009.12293*, 2020.

[48] Tianhe Yu, Deirdre Quillen, Zhanpeng He, Ryan Julian, Karol Hausman, Chelsea Finn, and Sergey Levine. Meta-world: A benchmark and evaluation for multi-task and meta reinforcement learning. In *Conference on Robot Learning*, pages 1094–1100. PMLR, 2020.

[49] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033, 2012.

[50] Franziska Meier, Evangelos Theodorou, and Stefan Schaal. Movement segmentation and recognition for imitation learning. In *Artificial Intelligence and Statistics*, pages 761–769. PMLR, 2012.

[51] Simon Manschitz, Michael Gienger, Jens Kober, and Jan Peters. Learning sequential force interaction skills. *Robotics*, 9(2):45, 2020.

[52] Tianhe Yu, Pieter Abbeel, Sergey Levine, and Chelsea Finn. One-shot hierarchical imitation learning of compound visuomotor tasks. *arXiv preprint arXiv:1810.11043*, 2018.

[53] Chen Sun, Abhinav Shrivastava, Saurabh Singh, and Abhinav Gupta. Revisiting unreasonable effectiveness of data in deep learning era. In *Proceedings of the IEEE international conference on computer vision*, pages 843–852, 2017.

[54] Suhua Lei, Huan Zhang, Ke Wang, and Zhendong Su. How training data affect the accuracy and robustness of neural networks for image classification. 2018.

[55] Xiang Long, Kaipeng Deng, Guanzhong Wang, Yang Zhang, Qingqing Dang, Yuan Gao, Hui Shen, Jianguo Ren, Shumin Han, Errui Ding, et al. Pp-yolo: An effective and efficient implementation of object detector. *arXiv preprint arXiv:2007.12099*, 2020.

[56] Jens Kober, J Andrew Bagnell, and Jan Peters. Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, 32(11):1238–1274, 2013.

[57] Hao nan Wang, Ning Liu, Yi yun Zhang, Da wei Feng, Feng Huang, Dong sheng Li, and Yi ming Zhang. Deep reinforcement learning: a survey, 12 2020.

[58] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[59] Harish Ravichandar, Athanasios S. Polydoros, Sonia Chernova, and Aude Billard. Recent advances in robot learning from demonstration. *Annual Review of Control, Robotics, and Autonomous Systems*, 3:297–330, 5 2020.

[60] Chao Yang, Xiaojian Ma, Wenbing Huang, Fuchun Sun, Huaping Liu, Junzhou Huang, and Chuang Gan. Imitation learning from observations by minimizing inverse dynamics disagreement. 10 2019.

[61] Snehal Jauhri, Carlos Celemin, and Jens Kober. Interactive imitation learning in state-space. 8 2020.

[62] Andrew Y Ng, Stuart J Russell, et al. Algorithms for inverse reinforcement learning. In *Icml*, volume 1, page 2, 2000.

[63] Tejas D Kulkarni, Karthik R Narasimhan, Ardavan Saeedi, and Joshua B Tenenbaum. Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. *arXiv preprint arXiv:1604.06057*, 2016.

[64] Andrew G Barto and Sridhar Mahadevan. Recent advances in hierarchical reinforcement learning. *Discrete event dynamic systems*, 13(1):41–77, 2003.

[65] Alexander Sasha Vezhnevets, Simon Osindero, Tom Schaul, Nicolas Heess, Max Jaderberg, David Silver, and Koray Kavukcuoglu. Feudal networks for hierarchical reinforcement learning. In *International Conference on Machine Learning*, pages 3540–3549. PMLR, 2017.

[66] Pierre-Luc Bacon, Jean Harb, and Doina Precup. The option-critic architecture. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 31, 2017.

[67] Luca Marzari, Ameya Pore, Diego Dall'Alba, Gerardo Aragon-Camarasa, Alessandro Farinelli, and Paolo Fiorini. Towards hierarchical task decomposition using deep reinforcement learning for pick and place subtasks. *arXiv preprint arXiv:2102.04022*, 2021.

[68] Hoang Le, Nan Jiang, Alekh Agarwal, Miroslav Dudík, Yisong Yue, and Hal Daumé. Hierarchical imitation and reinforcement learning. In *International Conference on Machine Learning*, pages 2917–2926. PMLR, 2018.

[69] Antonia Creswell, Tom White, Vincent Dumoulin, Kai Arulkumaran, Biswa Sengupta, and Anil A Bharath. Generative adversarial networks: An overview. *IEEE Signal Processing Magazine*, 35(1):53–65, 2018.

[70] Sergey Alatartsev, Sebastian Stellmacher, and Frank Ortmeier. Robotic task sequencing problem: A survey. *Journal of intelligent & robotic systems*, 80(2):279–298, 2015.

[71] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587, 2014.

[72] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. *Advances in neural information processing systems*, 28:91–99, 2015.

[73] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 2961–2969, 2017.

[74] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. Ssd: Single shot multibox detector. In *European conference on computer vision*, pages 21–37. Springer, 2016.

[75] Shijie Hao, Yuan Zhou, and Yanrong Guo. A brief survey on semantic segmentation with deep learning. *Neurocomputing*, 406:302–321, 2020.

[76] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015.

[77] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *IEEE transactions on pattern analysis and machine intelligence*, 39(12):2481–2495, 2017.

[78] Alberto Garcia-Garcia, Sergio Orts-Escolano, Sergiu Oprea, Victor Villena-Martinez, and Jose Garcia-Rodriguez. A review on deep learning techniques applied to semantic segmentation. *arXiv preprint arXiv:1704.06857*, 2017.