國立臺灣大學電機資訊學院電信工程學研究所

碩士論文

Department of Electrical Engineering
College of Electrical Engineering and Computer Science
National Taiwan University

Master Thesis

具注意力機制之混合任務級聯模型—實例分割之新框架

Hybrid Task Cascade with Attention: A New Framework for Instance Segmentation

Ricardo Manzanedo

指導教授：謝宏昀 博士

Advisor: Hung-Yun Hsieh, Ph.D.

中華民國 111 年 08 月

August, 2022

doi:10.6342/NTU202203163

國立臺灣大學碩士學位論文
口試委員會審定書

Master Thesis Certification by Oral Defense Committee
National Taiwan University

具注意力機制之混合任務級聯模型—
實例分割之新框架

Hybrid Task Cascade with Attention: A New Framework for Instance Segmentation

本論文係 Ricardo Manzanedo 君（學號 R08942139）在國立臺灣大學電信工程學研究所完成之碩士學位論文，於民國 111 年 8 月 30 日承下列考試委員審查通過及口試及格，特此證明

This is to certify that the Master thesis above is completed by Ricardo Manzanedo (ID R08942139) during his studying in Graduate Institute of Communication Engineering at National Taiwan University, and that the oral defense of this thesis is passed on 2022/08/30 in accordance with decision of following committee members:

口試委員 Committee members：

Hung-yu Hsu （指導教授/Advisor）

José Jesús Fraile Ardanuy Jesús Fraile

Firmado por JIMENEZ BERMEJO, DAVID (FIRMA) el día 30/08/2022 con un

系主任、所長(Department Chair/Program Director)

中文摘要

在需要能即時、有效檢測和物件分割的機器視覺任務，如自動駕駛汽車、行人跟蹤和接觸者追蹤檢測等，混合任務級聯 (HTC, Hybrid Task Cascade) 取得了里程碑的進展與諸多成功應用。近代的實例分割研究更多地集中在主幹模組、資料增強技術和基於轉換器的架構上。這些模型變得越來越複雜，因此也變得越來越慢。在本論文中，我們的目標是顯著降低所提出模型的大小和複雜性，但得同時保持 HTC 的最佳表現。與當前領先的實例分割技術相比，我們發現基於 HTC 架構中設計較無效率之處，因此，我們提出了帶注意機制的混合任務級聯 (HTCA) 框架，並在其中測試了三種不同的設計。在三種實驗設計中，將注意力機制嵌入反卷積層的混合任務級聯 (HTCA-D) 表現最好。HTCA-D 集成了最先進的檢測器 EfficientDet 作為主幹，為基於傳統 HTC 遮罩分割器的分支。分割任務也透過新模塊的合併能更聚焦於物件上。透過基準資料集的驗證比較，輕量化的 HTCA 不僅減少了使用的參數量，同時還能提高了目標偵測品質。使用 HTCA，我們在 COCO 資料集上增加 1.3 個遮罩 AP。

關鍵字：機器視覺、實例分割、混合任務級聯 (Hybrid Task Cascade)、注意力機制 (attention)、變換器 (transformer)

doi:10.6342/NTU202203163

ABSTRACT

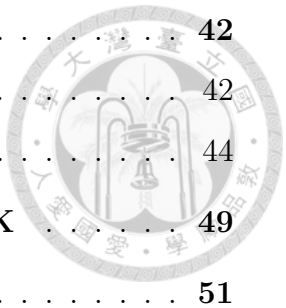
Hybrid Task Cascade (HTC) for instance segmentation has recently gained enormous interest in the computer vision community in the domains that require effective detection and segmentation in real-time, such as the self-driving car, pedestrian tracking, and contact tracing. Recent instance segmentation research focuses more on the backbone, data augmentation, and transformer-based architectures. These models became more and more complex and consequently slower. In this thesis, we aim to significantly reduce the size and complexity of the proposed model while maintaining the state-of-the-art performance of HTC. We have found inefficient designs in the previous HTC-based architectures compared to the leading-edge developments. Therefore, we tested three different designs in the proposed Hybrid Task Cascade with Attention (HTCA) framework. Among the three experimental designs, Hybrid Task Cascade with Attention in the deconvolutional layer (HTCA-D) appears to be the best, the proposed HTCA-D is a novel network that integrates the state-of-the-art detector EfficientDet as the backbone, followed by the segmentation branch based on the conventional HTC mask head. The segmentation task is also renovated by incorporating a new module to focus more on the object. Our method reduces the number of FLOPs by more than 30% and it uses almost 75% less memory than the original version of HTC. It helps to save time and energy consumption during training and inference. Through validation with benchmark datasets, the lightweight HTCA not only reduces the number of parameters used but also enhances the object detection quality at the same time. Using HTCA, we surpass our baseline mask AP and bounding box AP by +1.3 points in each task on the COCO dataset.

Keywords— Instance Segmentation, Hybrid Task Cascade (HTC), Attention

doi:10.6342/NTU202203163

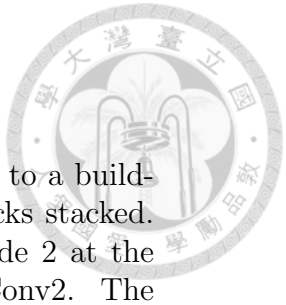
TABLE OF CONTENTS

doi:10.6342/NTU202203163



doi:10.6342/NTU202203163

LIST OF TABLES



doi:10.6342/NTU202203163

LIST OF FIGURES

doi:10.6342/NTU202203163

doi:10.6342/NTU202203163

doi:10.6342/NTU202203163

CHAPTER 1

INTRODUCTION

With the growth of multimedia data such as pictures, videos and so on, the development of automatic processing of these kinds of data became an important issue. Computer vision is a field which includes multiple tasks such as acquiring, processing, analyzing and understanding multimedia data, and extracting high-dimensional data from the real world. This field has drawn the attention of academic research and commercial application.

Towards the trend, the computer vision community has been emphasizing in several tasks such as object detection. However, other tasks such as instance segmentation do not perform that well yet. This task is a major task in the computer vision field which performs per-pixel classification of objects at the instance level. In this research area, instance refers to each individual object. Instance segmentation plays a crucial role in real applications such as self-driving vehicles and video surveillance.

Even though, astonishing achievements have been made in this area in recent years, approaching more accurate results. Most of the improvements have led to a high increment in the complexity of the models as well as the computational cost which ends up demanding more capacity and more energy. These factors are fundamental to discourage the deployment of most of the latest models in real-world scenarios where the capacity and the latency are very limited.

According to our investigation, Convolution Neural Network (CNN) architectures are the most common for instance segmentation task. It was in 2018, when Mask R-CNN [1] captivates all the computer vision community for its simple and flexible architecture. Since its release, there were some papers [2, 3] that have enhanced the performance by focusing on the mask head architecture. However, with the proposal of Hybrid Task Cascade for instance segmentation (HTC) [3] the researchers have changed their approaches to tackle this issue.

After the HTC was released, most of the developments achieved in the instance segmentation area were related either to the enhancement of the backbone architecture [4, 5, 6] or the implementation of new data augmentation techniques during training [7, 8, 9]. On the one hand, the improvements that came from the modifications of the backbones led to a raised in the complexity of the model. This rise causes a soar in the energy consumption needed since the hardware required to run these models needs to be more powerful. Indeed, these models also require

doi:10.6342/NTU202203163

more running time than previous models. All these facts make recent models to be hard to deploy in real-world scenarios.

On the other hand, the progress achieved with the data augmentation techniques also brings similar problems as the previous models. These proposals require more memory during the training period which led to higher power consumption. They also induce an increment in the training time need it to converge the model.

In recent years, with the introduction of the Tranformer [10] architecture, some researchers have followed this design to enhance previous instance segmentation results. Nonetheless, this architecture is more complex compared with the CNN-based models. Moreover, the transformer-based models require a higher training schedule to converge which also escalates the energy consumption. Hence, these models are not suitable for real-world applications.

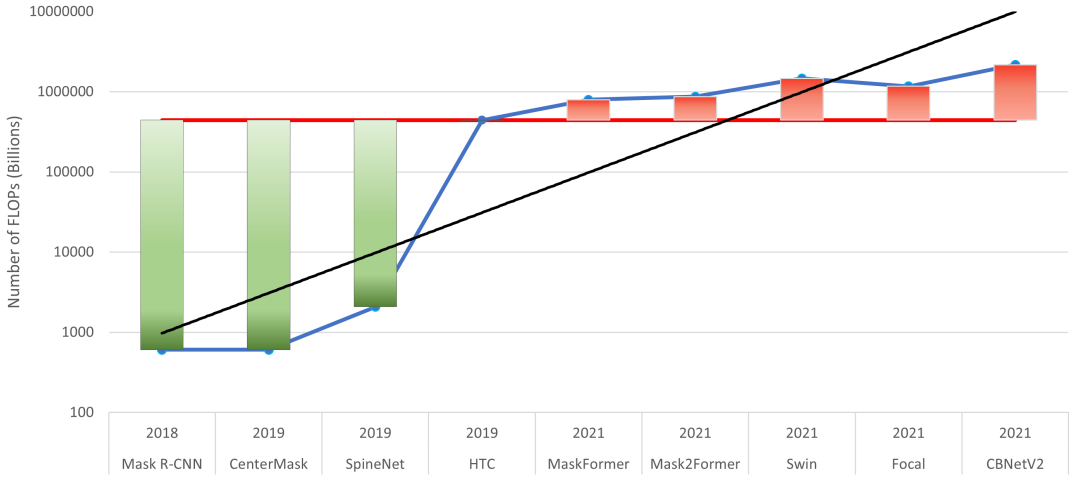


Figure 1: Evolution of Number of FLOPs in instance segmentation models. Models are sorted in terms of model precision. The red line Horizontal line represents the number of parameters of the HTC model. The black line denotes the tendency of the number of parameters. The blue line refers to the number of FLOPS each model has. The green and red bars represent a reduction and a soar in the number of FLOPs respecting the HTC model, respectively.

As illustrated in Figure 1, the trend of instance segmentation models is to use a higher number of floating point operations (FLOPs). This implies a rise in the computation complexity which can be translated into a higher demand for computational power and time. The Figure starts with the Mask R-CNN and HTC is the last model CNN-based. From 2021, all the models have at least one component transformer-based.

According to Figure 1, it seems that we need a larger number of FLOPS in order to obtain a better performance. There is only one exception which gets better performance with less complexity. The Focal model [11] is able to enhance its

doi:10.6342/NTU202203163

performance with data augmentation while training. In our related work analysis, we found that new research stopped focusing on CNN-based architectures. Even though, these models need less time to converge and they are less complex. These problems make new models hard to be implemented in real applications.

Although better performances have been achieved in recent years, most of the state-of-the-art models still use the HTC mask head to generate the instance segmentation. Indeed, the 80% of the top 10 uses this subnet to make the mask prediction. A natural question is:*Is it possible to enhance this subnet to improve the final precision?*

In this thesis, we are going to show how we can optimize some of these models by replacing some components of the state-of-the-art models with more efficient modules in terms of time and computational complexity. In addition, we analyze three architectures based on the well-known Hybrid Task Cascade with extra attention modules. We propose HTCA-D which incorporates a spatial attention module in each stage of the mask head to help the model to highlight the important features of the objects.

The rest of this thesis is organized as follows: Chapter 2 introduces the background knowledge including the concept of object detection and segmentation, and related works. Chapter 3 introduces multiple mask branches we implemented. Chapter 4 describes the proposed algorithms. Chapter 5 presents the results and performance evaluation of the proposed algorithms. Finally, Chapter 6 concludes this thesis.

doi:10.6342/NTU202203163

CHAPTER 2

BACKGROUND AND RELATED WORK

In this chapter, we refer to the background knowledge and related work to provide an overview of the current situation on the thesis´s topic. The background introduces segmentation tasks and the architecture of an instance segmentation model, while the related work summarizes the previous contributions on the multiple parts that compound such an algorithm.

2.1 Introduction to Segmentation

Image segmentation is the process of partitioning an image into multiple segments. We can see a clear example of this task in Figure 2. This task classifies each object corresponding to each group of pixels in the picture. There are two types of image segmentation: semantic segmentation and instance segmentation.

Figure 2: Image segmentation example [12].

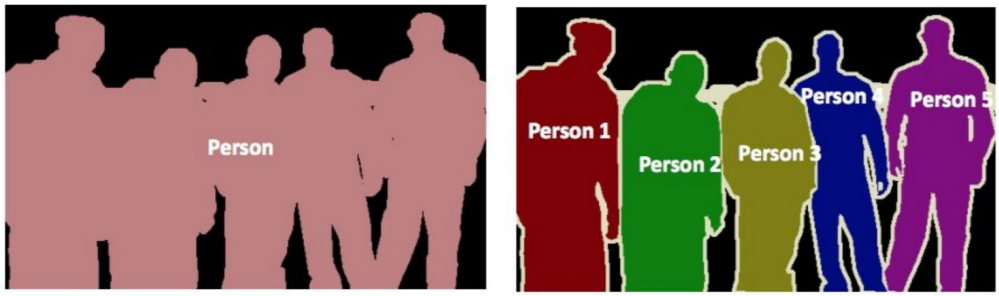
Semantic segmentation is a computer vision task where every pixel in the input image belongs to a class. And all pixels belonging to the same class are assigned to the same single color as we can see in Figure 3a. This kind of segmentation is usually approached by Fully Convolutional Networks (FCN) [13].

Instance segmentation is another kind of image segmentation where each instance of the same class is segmented individually. We can refer to Figure 3b for different instances of the same class. The segmentation of each instance is known as the mask of the object.

There are two main approaches to getting the instance segmentation. The first one would be by applying object detection first, and once we have all the

doi:10.6342/NTU202203163

objects proposed by the object detector we get the segmentation of the different instances, known as detection-based. The second method is to do the semantic segmentation over the input image and then detect the different objects in the image to distinguish the multiple instances, known as segmentation-based. Until 2021 detection-based methods achieved the best performance when in [14] proposed to use an old approach, mask classification which was used before FCN [13] was proposed, to tackle this task with a transformer-based architecture called MaskFormer [14].



(a) Semantic segmentation (b) Instance Segmentation

Figure 3: Two types of image segmentation [15].

In this thesis, we focus on detector-based instance segmentation. Therefore, we will introduce explaining the general architecture of an instance segmentation model.

2.2 General Architecture of Instance Segmentation

Nowadays, most instance segmentation models share a similar architecture which is composed of two main parts as we can see in Figure 4. The first part, it is commonly used in all computer vision tasks. It is formed by the backbone and the neck. The second part would be the different heads, bounding box head, mask head and classification head.

The initial part is the backbone usually pre-trained on ImageNet dataset [16], and it is the module that is fed by the input image. This backbone is formed by the sequence of several CNN that reduce the resolution of the input by several scales and it is in charge to extract the different features of the pictures which will be processed to detect the objects from the input image and generate the masks of each object, these features are called feature maps. In addition, the backbone is also known as the encoder of the model because it transforms the input image into the feature maps.

Afterwards, the backbones have moved from a single-size output to a Multi-Scale feature output in order to improve the model performance. This means

doi:10.6342/NTU202203163

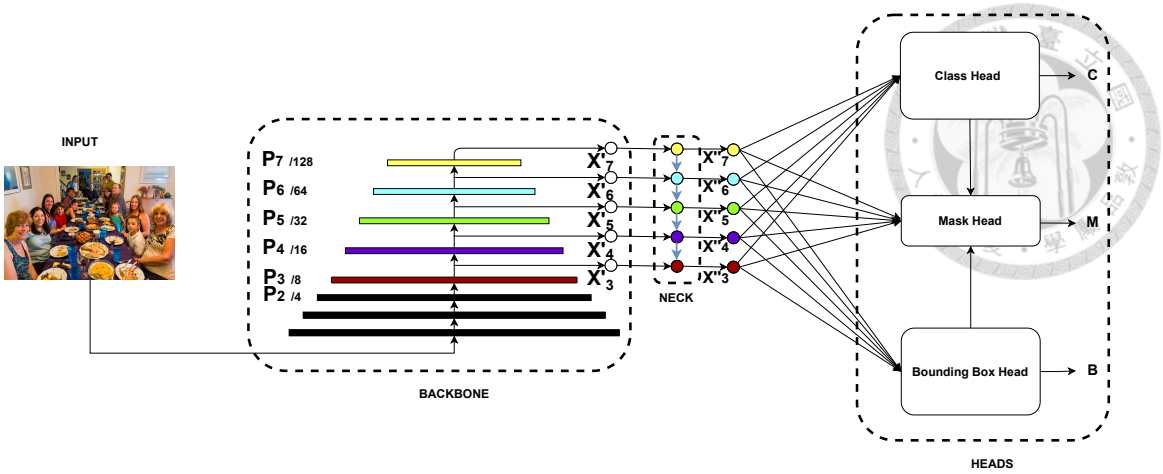


Figure 4: General architecture of computer vision tasks.

that recent models not only use the last feature maps generated by the backbone, instead they also take advantage of intermediate feature maps produced in the backbone which leads to creating a multi-scale feature output. This transformation was possible to the Feature Pyramid Network (FPN) [17] which is one of the pioneers to propose a combination of multi-scale features in 2016.

This multi-scale architecture has proved the importance of taking the features from different scales of the pictures and fusing them. Lately, the recent research for this module has moved from CNN to a transformer which is a new network architecture first presented by [18] in the Natural Language Processing (NLP) field, and then applied in the computer vision area by Google Brain Team [19].

Following a backbone with multi-scale output, there is a neck. The neck is a combination of bottom-up and top-down paths which helps to aggregate the different scales of information. Some of the necks that are often used in practice are FPN [17] or PANet [20]. The most frequent techniques used in this module are a series of convolutional networks or interpolation functions in order to set all the feature maps into the same scale and sum them up. It is demonstrated in several papers that this module helps to enhance the precision of computer vision models. The scale of these feature maps is usually a part of the input image and it is represented by Pi where $i \in [2, 7]$. This means that if we refer to Pi feature map, this feature map has a resolution of $\frac{1}{2^i}$ of the input.

As we mentioned above, the last part of the model is formed by the different heads; The bounding box head is the branch which generates the bounding box predictions. The classification branch is in charge to predict the class of the bounding boxes. Moreover, the computer vision models that only have these heads are called object detectors. There are multiple policies for classifying these models. The two most relevant rules to sort object detectors are: if the detector

doi:10.6342/NTU202203163

is anchor based, which is a method proposed in [21], or the number of stages that compose the detector.

The anchor-based detector uses anchors which is each hand-picked box of different height/width ratios that helps to identify objects of different shapes. Then, they classify if these boxes have or not an object within them before doing the bounding box regression and classification to identify the object class. Meanwhile, anchor-free detectors do not require the hand-picked boxes to perform the boxes regression and classification.

An object detector can be formed in one or two-stage. One-stage detectors only have a dense prediction module which is in charge of generating the Regions of Interest (RoIs). The RoIs are the parts of the image in which it could be an object. Two-stage detectors have a dense prediction module and a sparse prediction unit. The last unit is responsible to crop the RoIs and extracting the features from the RoIs to generate the final bounding box predictions. In addition, one-stage detectors use features from $P3$ (2^3) to $P7$ (2^7) that are larger receptive fields with lower-resolution, meanwhile two- stage detectors use feature levels of $P2$ (stride of 2^2) to $P5$ (2^5) .

The last head we will explain in the background is the mask head. This branch is usually formed by a sequence of CNN and FCN to classify each pixel of the object. However, new researches [14, 22, 23] are using transformer architecture for this head. As mentioned above we focus on detector-based models. Hence, we will explain the steps inside this type of model. In order to generate the mask predictions, these models required the bounding box predictions to crop these areas from the feature maps created by the neck. This process is done by applying RoIPool, RoIWrap or RoIAlign function. Once all the RoIs from the images are cropped, they are sent to a Convolutional Network that generates the masks of each RoI and each class. Then, the final mask predictions are selected by the label prediction of each RoI.

2.3 Related work

The rise of computing power has prompted researchers to investigate challenging issues in multiple areas. Instance Segmentation is a fundamental yet challenging computer vision task that the research community is paying high attention to. To present recent progress, we summarize recent research related to the different parts that compound a model for this kind of task.

doi:10.6342/NTU202203163

2.3.1 Backbones

Recently, there is an increment of interest from the Computer vision community in the optimization of backbones, also known as encoders. This Networks usually required a high number of parameters and computation.

2.3.1.1 ResNet

One of the common use models that encodes an input image is Residual Neural Network [24] (ResNet). [24] proposed this architecture in 2015 to solve the vanishing grading problems which previous models showed with the increment in the number of layers in the model. In their research, they demonstrated how these problems were mitigated by adding a shortcut connection to the architecture that would add the previous input X to the output $F(X)$ after a few blocks. A ResNet shortcut Connection is shown in Figure 5.

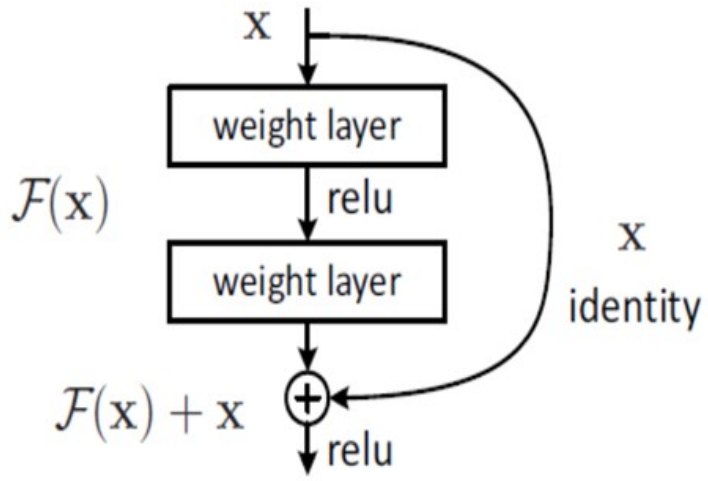


Figure 5: ResNet shortcut connection [24]. The X denotes the feature maps outputs from the previous layers. The weight layers are CNN and relu refers to an activation function. we can see the formal Eq. form in (2.1).

They implement this building block to prove that it is easier to optimize the residual mapping than to optimize the original, unreferenced mapping. This means that they denoted $\mathcal{H}(X)$ as the desired mapping, and let the stacked layers fit another mapping, $\mathcal{F}(X) = \mathcal{H}(X) - X$. Therefore, they obtained the desired mapping $\mathcal{H}(X)$ by adding the output mapping $\mathcal{F}(X)$ to the initial input X.

Formally, [24] defines the building block as:

$$\mathcal{H}(X) = \mathcal{F}(X, \{W_i\}) + X, \tag{2.1}$$

where X is the input and $\mathcal{H}(X)$ is the output of the shortcut. $\mathcal{F}(X, \{W_i\})$ represents the residual mapping to be learnt by the block. For example, in the Figure 5

doi:10.6342/NTU202203163

the output of the second weight layer would be defined as

$$\mathcal{F}(X, \{W_1, W_2\}) = W_2\sigma(W_1X), \tag{2.2}$$

where the biased of both layers have been omitted and σ denotes Rectified Linear Unit (ReLU).

ResNet can be seen as a family of encoders since there are different types of ResNet with similar architecture but a different number of layers. In this research, we focus on ResNet-50 which has three different types of configurations of layers inside each building block. The network use 1×1 kernel convolution layers at the start and end of each shortcut which reduce and increase the dimensions. The 3×3 layers are a bottleneck with smaller dimensions. A Layer-wise architecture design of ResNet-50 is shown in Table 1.

Table 1: Layers of the ResNet-50[24]. Each bracket corresponds to a building block, and the integer refers to the number of blocks stacked. Furthermore, there is a downsampling function of stride 2 at the beginning of Conv3, Conv4 and Conv 5, similar to Conv2. The output size is based on a 224×224 input image.

Layer Name	Output Size	50-layer
Conv1	112×112	$7 \times 7, 64$, stride 2
Conv2_x	56×56	3×3 max pool, stride 2
		$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$
Conv3_x	28×28	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 4$
Conv4_x	14×14	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 6$
Conv5_x	7×7	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$
	1×1	average pool, 1000-d fc, softmax
FLOPs		3.8×10^9

doi:10.6342/NTU202203163

Furthermore, this architecture allows us to generate feature maps in multiple sizes, at the end of each type of layer which has been proven that improve the performance in multiple tasks such as object detection and instance segmentation. As we could observe in Table 1, this network is a scale-decrease network.

2.3.1.2 SpineNet

SpineNet is a new family of encoders proposed by [25] in 2020. This network introduces a backbone with a new scale-permuted intermediate features and cross-scale connections. The way they implemented is with a list of N Building blocks $\{B_1, B_2, ..., B_N\}$ similar to the ResNet's Blocks. Each Block B_k is linked to a feature level L_i. The architecture of the blocks on the same level is identical. Moreover, they defined a multi-scale output with 5 levels (from L_3 to L_7). The level of the block represents the resolution of the feature map at that level, which means that the feature maps in level i have a resolution of $\frac{1}{2^i}$ the input image.

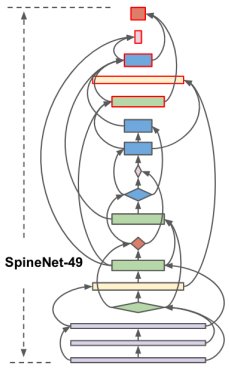


Figure 6: SpineNet-49 architecture [25]. The width of the block indicates the feature map resolution and the height denotes the number of channels. the rectangular blocks are the bottleneck blocks and the diamonds represent the residual blocks. Each color refers to one level and the output blocks have a red border.

2.3.1.3 EfficientNet

EfficientNet is one of the latest algorithms which improved accuracy and efficiency. This network fully concentrates on scaling. In [26] is demonstrated that this scaling generally improves the model accuracy with modification in depth, width and resolution scaling dimensions of CNN. **Depth** is about the number of layers in the model, **Width** explains the number of channels in the Convolutional layer and Resolution is about the input image **Resolution** that is passed to the CNN.

Different types of scaling including width scaling, depth scaling and resolution scaling are used for CNN network scaling but the compound scaling method

doi:10.6342/NTU202203163

(shown in Figure 7) is proposed in EfficientNet architecture which uniformly scales all three dimensions.

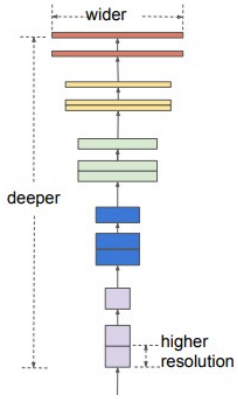


Figure 7: EfficientNet Compound Scaling [26]. The different colors represent the multiple levels of the model.

Table 2: Baseline network of EfficientNet-B0 layers [27]. Each stage i has a number of layers L_i with an input resolution of $H_i \times W_i$ and an output number of channels of C_i.

Stage i	**Operator** F_i	**Resolution** $H_i \times W_i$	**Channels** C_i	**Layers** L_i
1	conv3 × 3	224 ×224	32	1
2	MBConv1, K3 × 3	112 ×112	16	1
3	MBConv6, K3 × 3	112 ×112	24	2
4	MBConv6, K3 × 3	56 × 56	40	2
5	MBConv6, K3 × 3	28 × 28	80	3
6	MBConv6, K3 × 3	14 × 14	112	3
7	MBConv6, K3 × 3	14 × 14	192	4
8	MBConv6, K3 × 3	7 × 7	320	1
9	Conv1 × 1 & Pooling	7 × 7	1280	1

Manual scaling is possible but most of the time it reduces the accuracy and efficiency of the model. Actually, it is necessary to increase the depth and width of the network as there is a need to increase the image resolution. But it is a critical and challenging task to balance all the dimensions of the network therefore compound scaling play an important role to scale all the dimension and leads the network toward improved accuracy and efficiency. First, in [27] is defined as a good small baseline network and then they scale it along different dimensions creating a new family of backbones with different configurations from EfficientNet-B0 to EfficientNet-B6. EfficientNet architecture is similar to MnasNeT [28] and

doi:10.6342/NTU202203163

its baseline network is obtained by applying a Neural Architecture Search [29]. Network layers of EfficientNet-B0 details are shown in Table 2.

The MBConv in the network is an Inverted Residual Block that is mostly used in MobileNetV2 [30] architecture.

2.3.1.4 Vision Transformer

As mentioned in the previous Section the computer vision community is moving to transformer-based architectures, in [19] is proposed the Vision Transformer (ViT) which is the first transformer-based model used in computer vision. They show that the reliance on CNNs is not necessary and a pure transformer applied directly to sequences of image patches can perform very well as a backbone for a computer vision task. Even Though, they only implemented an image classification algorithm.

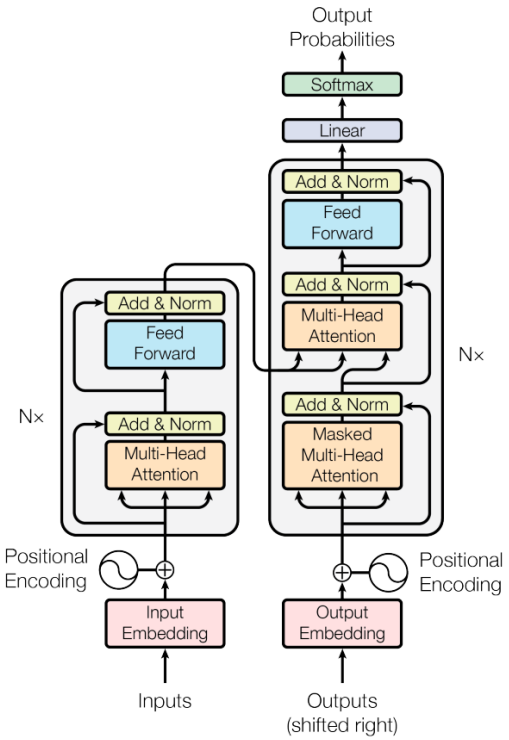


Figure 8: Transformer architecture [18]. The box on the left represents the encoder and on the right the decoder.

A transformer architecture is a compound of two main modules, one called the encoder and the other one known as the decoder. usually, the encoder is formed by a stack of 6 identical layers and each of these layers has two main sub-modules, a multi-head self-attention and a fully connected feed-forward. These sub-modules have a residual connection similar to the one used in CNN [31].

The decoder also consists of a stack of 6 layers. These layers have the same two

doi:10.6342/NTU202203163

sub-modules as the encoder plus an extra sub-module which is in charge to apply multi-head attention to the output of the last encoder. Same as in the encoder architecture it has residual connections in all the sub-modules. For more information related to this architecture refer to the transformer initial proposal [10].

An image patch in computer vision refers to a group of pixels of an image. This means that if an image divides into 16×16 patches then the image will be formed by 256 patches. For the ViT model, each of these patches will be taken as a token for the transformer module.

Furthermore, this article shows how this kind of model starts with lower performance than CNN models, however, if we use a large dataset to pretrain it, it can achieve better performance than recent state-of-the-art models.

2.3.1.5 Composite Backbone Network Version 2

In 2021, [5] proposed a combination of multiple pre-trained backbones as well as a new training technique for object detection. The CBNetV2 consists of K identical backbones ($K \geq 2$). The CBNetV2 architecture includes two types of backbones: the lead backbone B_K and assisting backbones $B_1, B_2, ..., B_{K-1}$ For the object detection task, only the output features of the lead backbone $\{x_K^i, i = 2, 3, ..., L\}$ are fed into the neck and then the RPN/detection head. They propose to generate initial results of assisting backbones by supervision with the auxiliary neck and detection head to provide additional regularization, as shown in Figure 9.

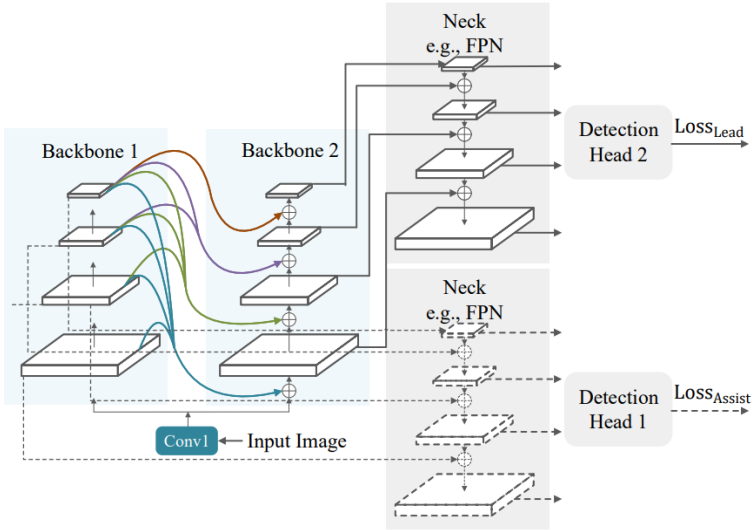


Figure 9: CBNetV2 (K=2) training structure with assistant supervision [5]. Both necks and detection heads use the same weights.

doi:10.6342/NTU202203163

2.3.1.6 Swin Transformer

In [4] is presented a new vision Transformer, called Swin Transformer, that capably serves as a general-purpose backbone for computer vision tasks.

They propose Swin Transformer, a new model based on transformers which produces a hierarchical feature representation similar to the Multi-scale feature maps generated by the traditional CNNs and has linear computational complexity with respect to the input image size.

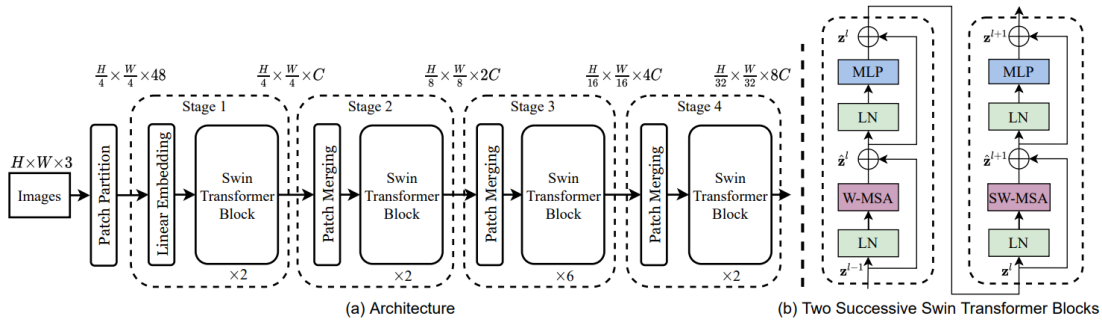


Figure 10: (a) Swin Transformer (Swin-T) architecture; (b) two successive Swin Transformer Blocks. W-MSA and SW-MSA are multi-head self-attention modules with regular configuration as present in ViT [19] and shifted windowing structure proposed by Liu [4].

Furthermore, they also focus on the fact that the window-based self-attention module lacks connections across windows, which limits its modeling power. To introduce cross-window connections while maintaining the efficient computation of non-overlapping windows, they propose a shifted window partitioning approach which alternates between two partitioning configurations in consecutive Swin Transformer blocks.

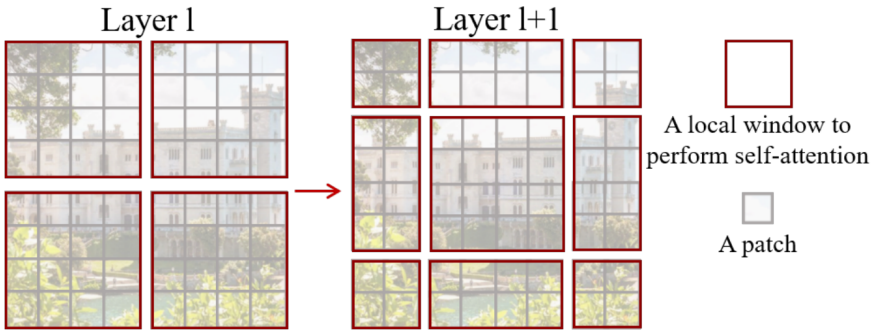
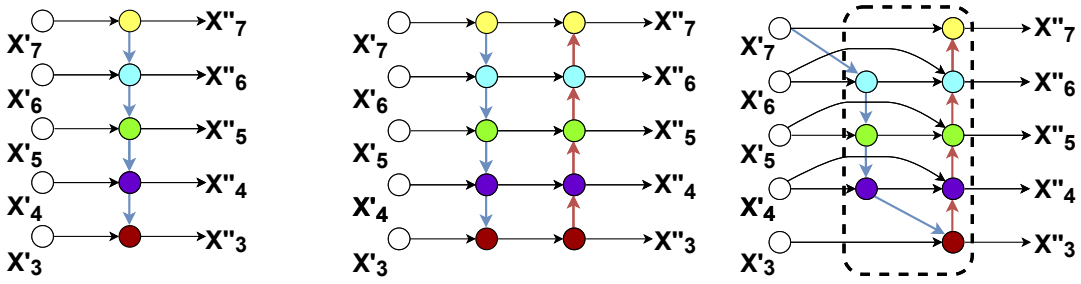


Figure 11: Shifted Window approach for computing self-attention architecture. On the left layer l, self-attention is computed within each regular partitioning window. On the right layer $l + 1$, the window partitioning is shifted, resulting in new windows [4].

doi:10.6342/NTU202203163

2.3.2 Necks

Recognizing objects at extremely different scales is a crucial computer vision challenge. With the development of the multi-scale output backbone, the importance of the neck used in the models has increased in order to obtain stronger feature maps. In this section, we explain some of the most relevant necks in this field.



(a) FPN (b) PANet (c) BiFPN

Figure 12: Feature network design. (a)FPN with a top-down path where blue arrows refer to the up-sampling function. The Eq. can be found in (2.3). (b) PANet with an extra bottom-up path where red arrows represent the down-sampling function. The Eq. form of the bottom-up path is (2.5). (c) BiFPN layer, a more efficient neck. All these subnetworks fuse multi-scale output from level $P3$ to level $P7$. we can appreciate an example of the mathematical expression of level $P4$ on Eq. (2.7) and (2.8).

2.3.2.1 Feature Pyramid Network

As mentioned above FPN is one of the pioneers in this kind of module. In 2017, [32] propose a convolutional FPN. This FPN consists of a top-down path with lateral connections that fuse the different scales output of the backbone. In this paper, they also show that using multiple levels of feature maps to obtain the prediction can improve the performance of the model.

In Figure 12a, we can see how are the connections inside this network. First of all, the top-down path takes the higher feature levels and upsamples them to the prior feature map scale. This up-sampling function takes a scaling step of 2. Then, the resulting feature map is summed to the output of a lateral convolutional layer corresponding to the backbone output feature map. We can see in Equation (2.3),

$$X_i'' = Resize(X_{i+1}'') + Conv(X_i'), \tag{2.3}$$

where X_i'' refers to the output of level i of the FPN. $Resize$ represents the up-sampling function and $Conv$ is a 1×1 convolutional network. This equation is an element-wise addition.

doi:10.6342/NTU202203163

Furthermore, they proposed Equation(2.4) in order to select the right level feature maps for the extraction of the RoIs for each proposal of the different heads. They demonstrate that this technique can enhance the performance of different heads of the model.

$$k = \lfloor k_0 + \log_2(\sqrt{wh}/224) \rfloor, \tag{2.4}$$

where 224 is the canonical ImageNet pre-trained size [16], w and h are the width and the height of the RoI, respectively. k_0 is the target level for a RoI of $w \times h = 224^2$, which in this case is set to 4.

2.3.2.2 Path Aggregation Network

Following this trend, in [20] is suggested a modification in the FPN architecture. At the same time, they propose a new approach to tackling the selection of the feature map level for the extraction of the RoIs in the heads.

As we can appreciate in Figure 12b, the FPN modification made by [20] consists in adding an extra bottom-up path after the top-down path. The new connections of the bottom-up are the same as the top-down, except for the up-sampling function that is substituted by a down-sampling function. Therefore, the output of the new links follows Equation (2.5),

$$X_i'' = Resize(X_{i-1}'') + Conv(X_i') \tag{2.5}$$

where X_i'' represents the output of the bottom-up path and the *Resize* operation is a down-sampling function. X_i' refers to the feature map generated in the top-down path.

2.3.2.3 Bidirectional Feature Pyramid Network

As a neck structure, most of the previous networks use a top-down approach [17] that limited by is one-way information flow, Path Aggregation Network (PANet) [20] includes bottom-up information flow but it is computationally costly. A novel Bidirectional Feature Pyramid Network (BiFPN) is proposed in EfficientDet [27] where information flows in both top-down and bottom-up directions and it is efficient with low computational cost. All the nodes in this layer have at least two input edges to simplify the previous bidirectional network as shown in Figure 12c. Further, they add an extra edge from the original input to the output node at the same level. For feature fusion, a fast normalization fusion approach is proposed in order to reduce time consumption.

There is an issue with traditional processes because it is necessary to resize resolutions when fusing features with different resolutions. But it is observed

doi:10.6342/NTU202203163

that different input features are at different resolutions which can't contribute to output features equally. Therefore, the additional weight concept for each input which able the network to learn the importance of each input feature map. The BiFPN layer integrates bidirectional cross-scale connections and the fast-normalized fusion, which let the network learn the importance of each input. We can see in Equation (2.6) this fast-normalized fusion,

$$O = \sum_i \frac{w_i}{\epsilon + \sum_j w_j}. \tag{2.6}$$

As an example, we describe this method for the second level ($\mathbf{X}'_4$) in the follow Equations (2.7) and (2.8):

$$\mathbf{X}'^{td}_4 = Conv\left(\frac{w_1 \cdot \mathbf{X}'_4 + w_2 \cdot Resize(\mathbf{X}'_5)}{w_1 + w_2 + \epsilon}\right), and \tag{2.7}$$

$$\mathbf{X}''_4 = Conv\left(\frac{w'_1 \cdot \mathbf{X}'_4 + w'_2 \cdot \mathbf{X}'^{td}_4 + w'_3 \cdot Resize(\mathbf{X}''_3)}{w'_1 + w'_2 + w'_3 + \epsilon}\right), \tag{2.8}$$

where $\mathbf{X}'^{td}_4$ is the feature in the middle step before the output. Furthermore, we can observe in Figure 12c that the blue and red arrows represent down-sampling and up-sampling respectively. This operation is needed for matching the different resolutions.

2.3.3 Mask Head

In recent years, the focus on instance segmentation task has soared inside the computer vision community. This attention led to an improvement in the model's performance. Most of these improvements have come by optimizing the backbone. However, there are some researches that have focused on boosting the mask head part of the model.

2.3.3.1 Mask R-CNN

Mask R-CNN [1] is a well-Known object instance segmentation model that was proposed by the Facebook AI Research group. It is a simple and flexible framework which efficiently detects and generates high-quality masks for each detection.

In this section, we will focus only on the RoIAlign and the mask branch of this model because it is a flexible model that we can implement in our research. The RoIAlign layer removes previous quantization problems caused by RoIPool, it uses a bilinear interpolation to obtain the exact value of the feature map when it is crop and aggregate the output to an unified scale, eg. 7×7. This layer provides a huge improvement compared with other layers such as RoIWrap [33]

doi:10.6342/NTU202203163

or RoIPool [34]. In Figure 13, we can observe how does RoIAlign is applied to a feature map to calculate the values of the RoI.

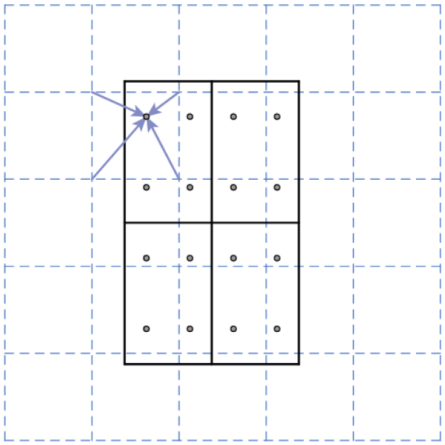


Figure 13: RoIAlign [1]. The blue grid represents the feature map and the black grid is the RoI area. RoIAlign computes each value of the dots inside the RoI by applying a bilinear interpolation of the closest grid points of the feature map.

On the other hand, there is the mask branch which is formed by a block of four convolutional layers, followed by a deconvolution and a final convolutional module, before a sigmoid function which will serve as an activation function. The architecture of this branch is shown in Table 3.

Table 3: Mask head architecture.

Layer	Kernel	Strides	Channels in	Channels out	Padding
Conv2d	3	1	# of input channels	256	(1,1)
Conv2d	3	1	256	256	(1,1)
Conv2d	3	1	256	256	(1,1)
Conv2d	3	1	256	256	(1,1)
Deconv2d	2	2	256	256	-
Conv2d	1	1	256	# of classes	-

Furthermore, [1] use a ResNet as backbone with a multi-scale output from level $P2$ to level $P5$, which means that the resolutions of the output feature maps go from $\frac{1}{2^2}$ to $\frac{1}{2^5}$ of the input image. Since they are using a backbone with multi-scale outputs, they also implement a FPN [17] as neck in order to fuse the backbone outputs and enhanced the performance. Following the FPN details, they also used Equation(2.4) in order to select the right level feature maps for the extraction of the RoIs for each proposal.

doi:10.6342/NTU202203163

2.3.3.2 Centermask

Here, we will take a closer look at the Spatial Attention Module (SAM) proposed in the CenterMask [2]. In this Network, it is suggested a new variant of the Mask branch from Mask R-CNN. They suggest the addition of a SAM between the first blocks of 4 convolutional layers and the deconvolution of the mask head. The SAM first generates pooled features p_{avg}, p_{max} and aggregates them via concatenation. Then it is followed by a 3×3 convolutional layer and normalized by the sigmoid function.

$$A_{sam}(X_i) = \sigma(\mathcal{F}_{3\times3}(p_{avg} \circ p_{max}), \tag{2.9}$$

where σ represents the sigmoid function, $\mathcal{F}_{3\times3}$ denotes a 3×3 convolution layer and $\circ$ is the concatenation operation.

Finally, they apply an element-wise multiplication of the input feature map, X_i, and the previous output, $A_{sam}(X_i)$ to focus on the important parts of the image. As we can see in the formula (2.10),

$$X_{sam} = A_{sam}(X_i) \otimes X_i, \tag{2.10}$$

where $\otimes$ represents the element-wise multiplication. The output of this module goes to the last two layers of the mask branch. [2] shows how a simple attention module can enhance the final mask prediction.

In addition to the SAM module, [2] also proposes a new Equation (2.11) for the mapping of the proposal to the output feature maps for the RoI extraction, so it does not depends on the canonical size of the ImageNet dataset.

$$k = \lceil k_{max} - \log_2(A_{input}/A_{RoI}) \rceil, \tag{2.11}$$

where k_{max} is the last level of the feature maps. A_{input} and A_{RoI} refers to the areas of the input image and RoI. The last important finding within this paper for this research is that [2] analyzes the performance of selecting a different range of feature maps to extract the RoI. They claim that their model achieves the best performance by selecting only the feature maps from $P3$ to $P5$. For more details about this topic please refer to CenterMask [2].

2.3.3.3 Hybrid Task Cascade

HTC was proposed by [3] in 2019, and it still being the instance segmentation architecture used for most of the state-of-the-art models. HTC generates the mask in parallel with the bounding boxes in the different stages, it allows to inter-connect the information of the bounding boxes with the mask information.

doi:10.6342/NTU202203163

HTC architecture uses a neck for the outputs of the backbone, which is an FPN that sets all the channels of the feature maps to 256 and adds an extra level of size $P6$. Then, these feature maps are used to generate a semantic segmentation map as well as the bounding box prediction and mask generation. On the one hand, the semantic branch is based on the FPN architecture, and it scales all the input feature maps to the same scale via a combination of 1×1 convolutional layers where the stride is set up to 8. at this moment, all the feature maps are fused by an element-wise sum. Furthermore, the result feature map goes to for 3×3 convolutional layers and one 1×1 convolution to obtain the semantic segmentation.

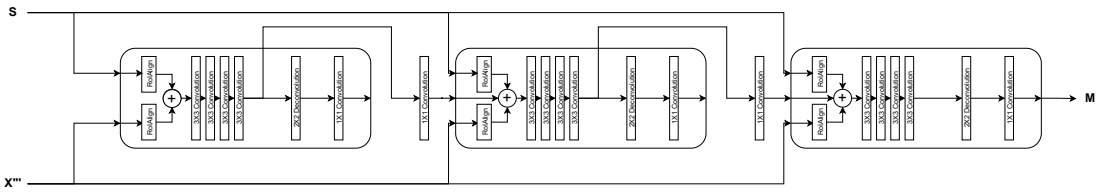


Figure 14: Mask flow implementation [3].

On the other hand, the outputs of the neck are employed as the inputs of the RPN which produce a list of bounding boxes proposal that will be used as the input of the first stage of the head box head. Then, the predictions of this head will be used to perform a RoIAlign operation to the corresponding level of the initial feature maps of this module and to the semantic map generated via the semantic branch, as we can see in Figure 14. These feature patches from both sources are element-wise sum in order to combine the information from the two sources. In the first stage of the mask head, the result from the previous addition is fed to four 3×3 convolutional layers similar to the architectures explained above. Later, these feature maps will be used for the mask generation of the first stage and they will be also passed to the second stage mask head. To create the mask predictions the output from the last convolutional layer goes to a deconvolutional layer and a 1×1 convolutional layer as in Section 3.3.1. Furthermore, before the feature maps generated by the first convolutional layers of the mask head in the first stage are added to the input of the mask head in the second stage, they go through a 1×1 convolutional layer.

2.3.3.4 MaskFormer

Following the current direction of the computer vision community of using a transformer architecture, the mask head is proposed in [14]. One thing new about this model compared with the previous ones is that it does not rely on the bounding boxes to generate the mask predictions.

doi:10.6342/NTU202203163

First of all, they use the outputs of the backbone as inputs of the pixel decoder which is an architecture similar to FPN and the output of stride 32 ($P5$) as input of the transformer decoder. The pixel decoder module is in charge to create a semantic segmentation, while the transformer decoder will propose N regions and classify them. At first, all the inputs of the pixel decoder are fed to a convolutional layer to generate feature maps with 256 channels on every scale. Then, the pixel decoder sends the feature Maps in stride 32 to the encoder part of the transformer module and it fused all scales top-down into a stride 4 ($P2$) using FPN architecture. This combination of all scales goes through a 3×3 convolutional layer to produce per-pixel embeddings of dimension 256.

The transformer decoder module has the standard architecture proposed by [10], which is compounded by 6 encoders and 6 decoders. The feature maps of stride 32 mentioned before together with a positional embedding are fed to the encoder part, which is 6 sequential encoders. The last output of this section with another positional embedding is used as inputs of the decoder module, which is a compound of 6 sequential decoders that each one produces N different regions proposals. From the decoder module, they use the output of each decoder to train the module. Then, these feature maps are sent to a linear layer to classify the N regions into different classes, and to the MLP layer in order to create the final N regions proposals. The final proposals and the per-pixel embeddings will be multiplied to get the final masks. Since it is a matrix multiplication all the predicted masks will have the same scale as the per-pixel embeddings which are stride 4.

2.3.3.5 Mask2Former

To conclude with this Chapter 2, we are going to explain the performance of a multi-scale transformer used in the mask head. Even though, this head was also proposed by [22] in 2021.

As well as in the model in Section 2.3.3.4, They use the backbone outputs as inputs of the pixel decoder. The pixel decoder used in this model is the most advanced multi-scale deformable attention Transformer (MSDeformAttn) [35]. This module is composed of 6 MSDeformAttn layers and fed with the feature maps from $P2$ to $P4$. The feature map of stride 8 is fed to upsampling layer and fused with the $P2$ feature maps to generate the per-pixel embedding of dimension 256.

Then the output of the MSDeformAttn module will go to the MultiScale Transformer decoder. This module is a compound of 3 different levels and each level has up to 3 transformer decoders, making a total of 9 layers. The key component of the transformer decoder is that it used a special masked attention operator in order to switch the position of the self- and cross-attention module. therefore, the

doi:10.6342/NTU202203163

attention matrix computation of the output would be as shown in Equation (2.12),

$$\mathbf{X}_l = softmax(\mathcal{M}_{l-1} + \mathbf{Q}_l\mathbf{K}_l^T)\mathbf{V}_l + \mathbf{X}_{l-1}, \tag{2.12}$$

where, l refers the index of the layer, $\mathbf{X}_l$ denotes to the query features at the l^{th} layer. $softmax(\mathcal{M}_{l-1}$ is the attention mask generated by the previous layer. $\mathbf{M}_0$ is the binary mask produced from $\mathbf{X}_0$, and $\mathbf{X}_0$ refers to the input of the transformer decoder module.

Similar to the MaskFormer, the output of the transformer decoder is used to obtain the class prediction as well as the final N proposal regions. the class predictions are produced by a linear layer and an MLP layer generates the final N regions which will be multiplied by the per-pixel embeddings to create the final mask predictions. Likewise, the MaskFormer article also used the output of each layer of the transformer decoder in order to train this module with the auxiliary losses.

doi:10.6342/NTU202203163

CHAPTER 3

MODELS

In this Chapter, we focus on the multiple models we have configured to compare their performance with the respective baselines using ResNet 50 as the backbone. We start selecting an optimal backbone and continue implementing the different mask heads explained in Section 2.3.3. Finally, we discuss the advantages and disadvantages of each configuration.

3.1 Motivation

We found that current baseline models in the computer vision field are based on ResNet [31] backbones with FPN. However, new backbone architectures have been proposed since 2015 when ResNet was presented for the first time. Other backbones had demonstrated a better performance than the baseline, but most of the time this improvement came with an increase in parameters and a raise in computational complexity. Both of these problems lead to soaring energy consumption and the time needed to process images, which is a huge problem to implement in real-world scenarios. A natural question is: *Is it possible to use a more optimal backbone in terms of computational complexity, obtaining higher accuracy than the current baselines?* This chapter aims to tackle this issue by studying the performance of an optimal backbone architecture with different mask heads and comparing it to their baseline models.

3.2 Backbone

As mentioned in Chapter 2, best instance segmentation models are object detection based. This is why we decide to first find an optimal object detector that has not been used for this task and compared it with a baseline architecture used by everyone for Instance segmentation, which is Mask R-CNN [1].

In 2020, in [27] proposed the most optimal object detector until today based on CNN architecture and it has not been used for instance segmentation task. This model is compound by an EfficientNet as the backbone, and the BiFPN as the neck that we explain in detail in Chapter 2 and, a class and a box subnets to generate the final predictions following the proposed architecture by [36].

As we explained in Chapter 2, EfficientNet has multiple configurations based on the input resolution as well as the depth of the EfficientNet. In [27] is also

doi:10.6342/NTU202203163

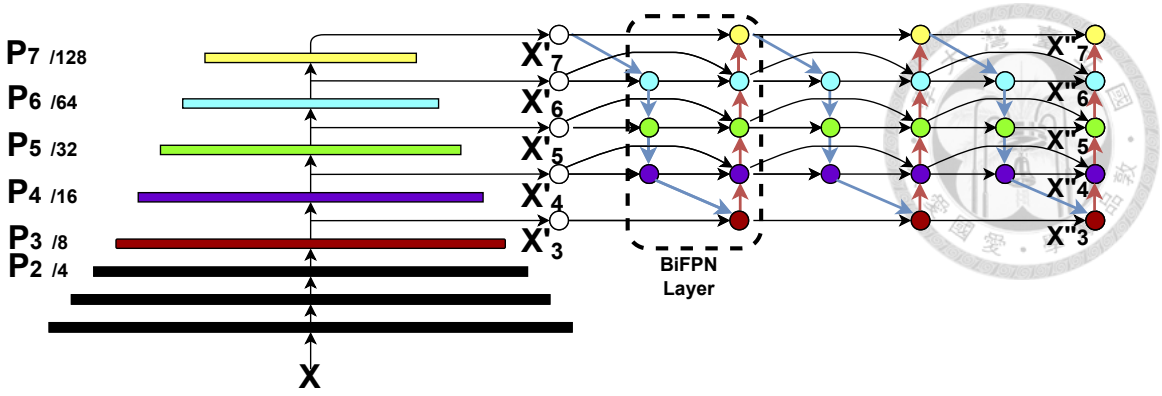


Figure 15: EfficientDet architecture [27].

proposed something similar with the whole architecture creating a new family of object detectors, in which all the network is scaled in depth, width and resolution. We can see the details of all the configurations in Table 4. Therefore, we need to make an analysis of these configurations to select the proper one for our research.

Table 4: Configutations of EfficientDet D0-D7 [27].

	Input Size R_{input}	Backbone Network	BiFPN #channels W_{bifpn}	BiFPN #layers D_{bifpn}	Box/Class #layers D_{class}
D0 ($\phi = 0$)	512	B0	64	3	3
D1 ($\phi = 1$)	640	B1	88	4	3
D2 ($\phi = 2$)	768	B2	112	5	3
D3 ($\phi = 3$)	896	B3	160	6	4
D4 ($\phi = 4$)	1024	B4	224	7	4
D5 ($\phi = 5$)	1280	B5	288	7	4
D6 ($\phi = 6$)	1280	B6	384	8	5
D7 ($\phi = 7$)	1536	B6	384	8	5

Due to the limitation of computational resources which we only count with one GPU of 12 GB, we decide to use Mask R-CNN with ResNet 50 and FPN to do the comparison of this research. Hence, we need to select an EfficientDet configuration that meets similar characteristics as the ResNet 50 with an FPN.

To make our decision, we make an analysis based on the number of parameters, number of Flops and inference time of all the EfficientDet configurations. The results of this analysis are shown in Figure 16.

According to the requirements mentioned above, we select the EfficientDet-d3 backbone and neck because it has a similar inference time and less complexity than the ResNet 50 with the FPN. Moreover, [27] use this configuration to compare their performance in the official EfficientDet article [27].

doi:10.6342/NTU202203163

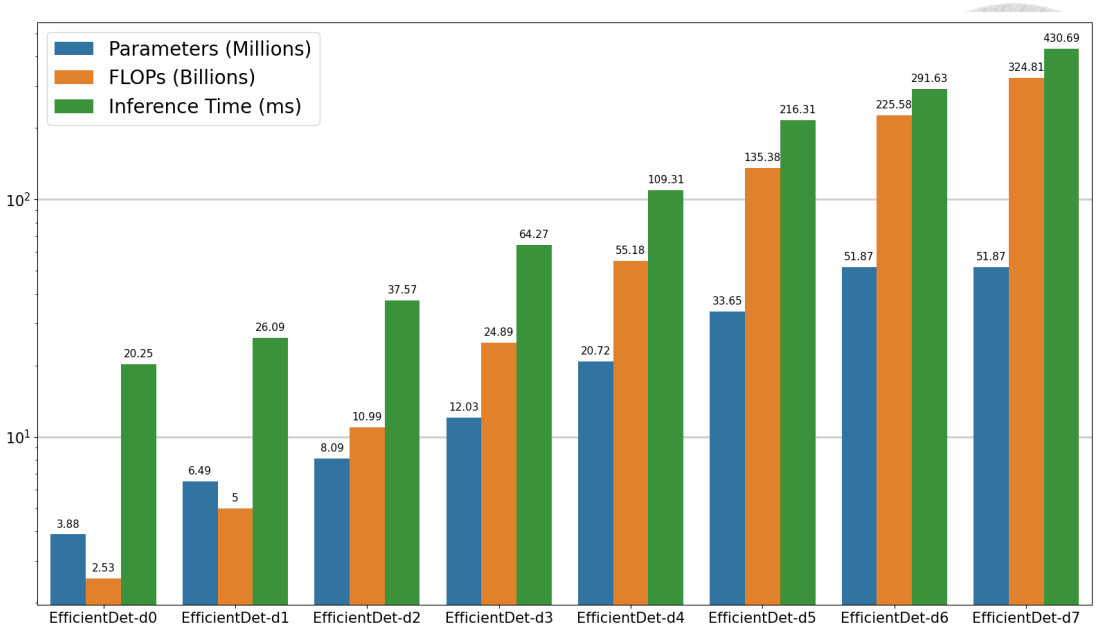


Figure 16: Comparison of all EfficientDet configurations, in terms of number of Parameters, number of FLOPs and Inference Time.

Furthermore, we make deeper insights in Figure 17 the complexity of our choice EfficientDet-d3 with the detection part of the Mask R-CNN which is the same as Faster R-CNN compound by ResNet 50 [31] with the FPN [17]. As we can observe, the number of parameters of the architecture we chose decreased by more than 50% and the number of FLOPs (floating point operations) shrink by almost 70%.

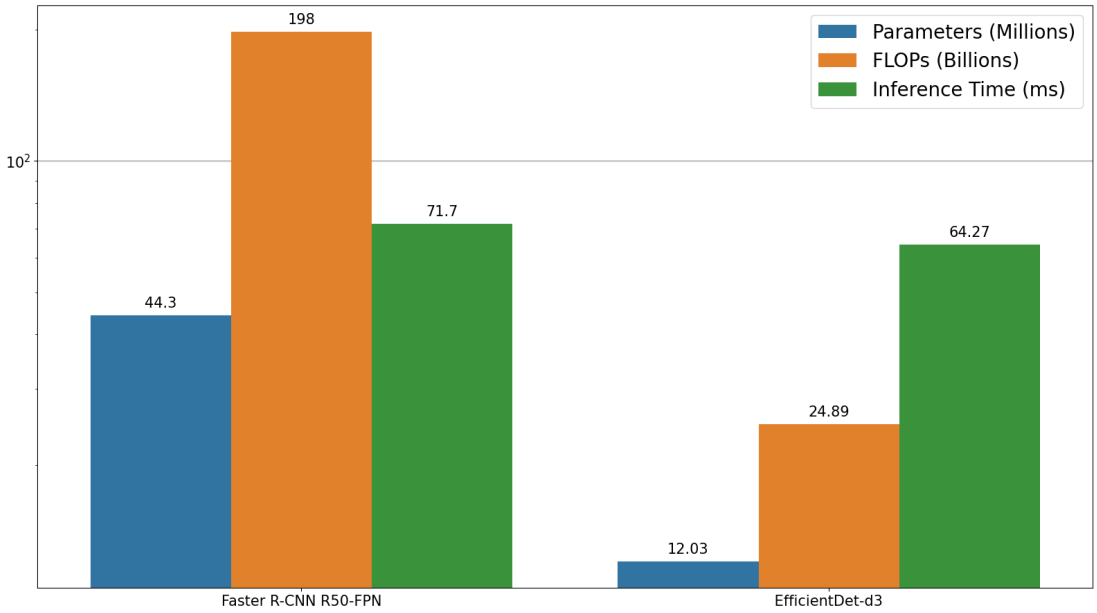
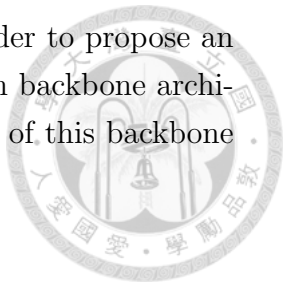


Figure 17: Comparison of the number of parameters, FLOPs and inference time of the backbones.

Due to Figure 17, we believe EfficientNet-d3 as the backbone with 6 layers of

doi:10.6342/NTU202203163

BiFPN as the neck is a great choice for our configuration, in order to propose an optimal instance segmentation model. Once we select a common backbone architecture for our research, we are going to study the performance of this backbone in the instance segmentation task using several mask branches.



3.3 Mask Branch

In this Section, we will introduce all the different architecture we have implemented in this research. We will start with the most simple and earliest networks, and continue with the most recent proposals.

3.3.1 Mask R-CNN

As mentioned in Chapter 2, this is a simple and flexible framework which generates a high-quality mask. In order to implement this branch to our backbone, we also need to use the classification branch and bounding box prediction head from EfficientDet [27] original proposal. The predictions combined with the feature maps need to be adapted to feed the RoI head module of the Mask R-CNN in which we only use the mask branch to generate the mask predictions. For this implementation, the author normalized the input data according to the mean and standard deviation of the ImageNet dataset. However, we are using the EfficientDet pre-trained model that does not require this normalization, in fact, it downgrades the model performances.

In addition, we have made use of the Equation (2.11) proposed by [2] in order to assign the feature levels to extract the RoIs for the mask head. Following, the same article of CenterMask [2] we use only the feature maps form $P3$ to $P5$ because EfficientDet has outputs from scale $P3$ to $P7$.

With this model, we expect to have a baseline model for this thesis. We select this branch for its simple implementation and its common use within the computer vision community in order to compare its results with the new proposals.

On the other hand, this approach is very light in terms of memory and complexity added to the EfficientDet. Since it is only compounded by 6 convolutional layers. According to our tests, the average time to process one image during the training is about 83.9 milliseconds(ms), it counts with 50.94 Billions of FLOPs and the average memory is about 67 MB for an image 896×896. In Figure 18, we can observe a comparison with the original model proposed in [1].

3.3.2 CenterMask

Following the trend of applying attention modules to help object detectors to focus on the important regions and eradicate the irrelevant features. We decided

doi:10.6342/NTU202203163

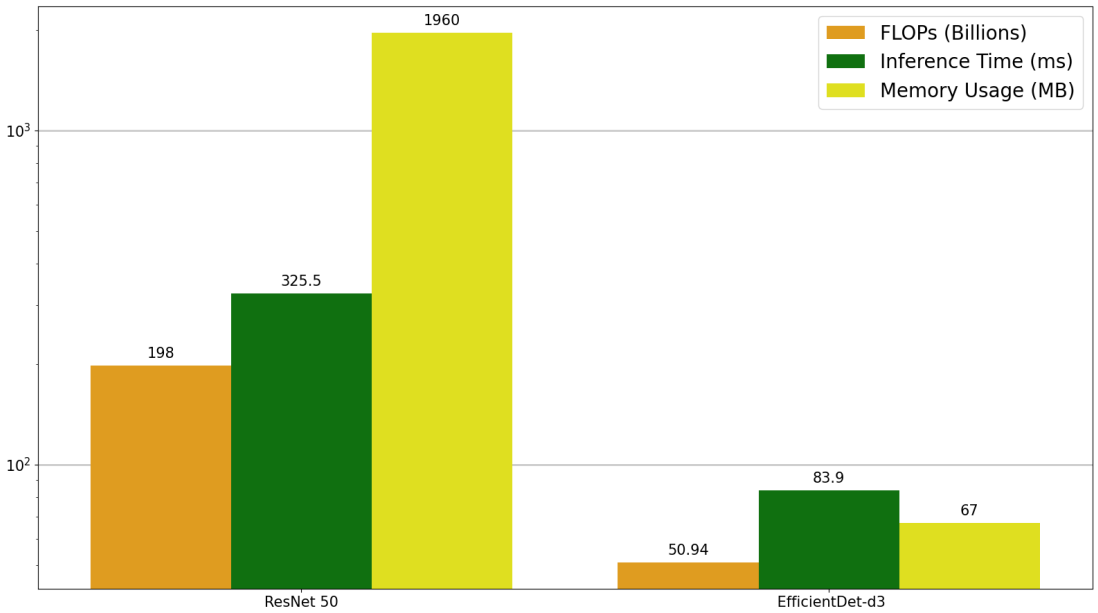


Figure 18: Comparison between Mask R-CNN with FPN and ResNet 50 and our baseline model compound EfficientDet-d3 and the mask head from Mask R-CNN [1].

to study the performance of the Spatial Attention-Guided Mask branch proposed by [2] with the backbone we have mentioned above.

The implementation of this branch consists only in adding a SAM, to the preceding Mask r-cnn branch. Also, we change the level assign equation and the features fed to the mask branch following the proposal made in [2]. we hope to improve the performance of our baseline model 3.3.1, with a model that does not add relevant complexity to our previous model. Our hypothesis is based on the fact that this SAM module helps the mask branch to focus on the object's silhouette.

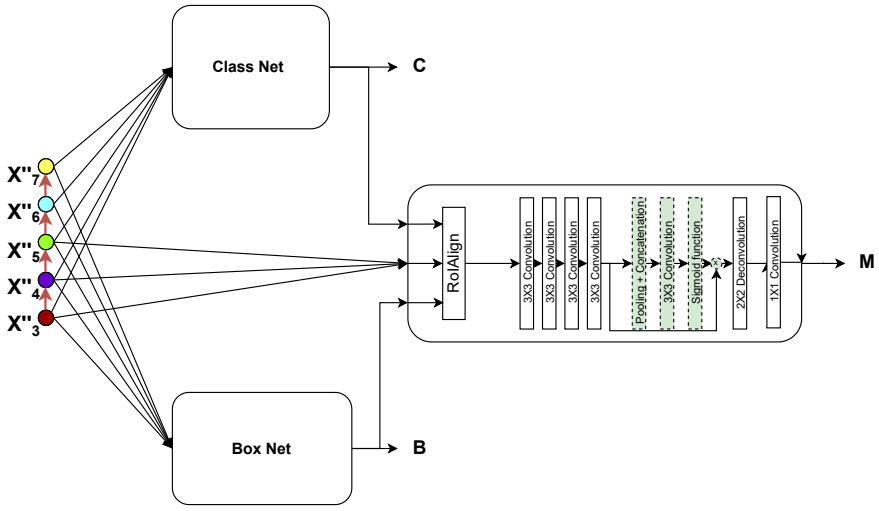


Figure 19: Mask head of the Centermask [2].

doi:10.6342/NTU202203163

In Figure 20, it is shown a comparison in terms of computational complexity, the time required by iteration and GPU memory needed to train per image of size 896×896. As we can see, the number of FLOPs is reduced by more than 75% as well as the time needed, and the GPU memory used is decreased by 96%.

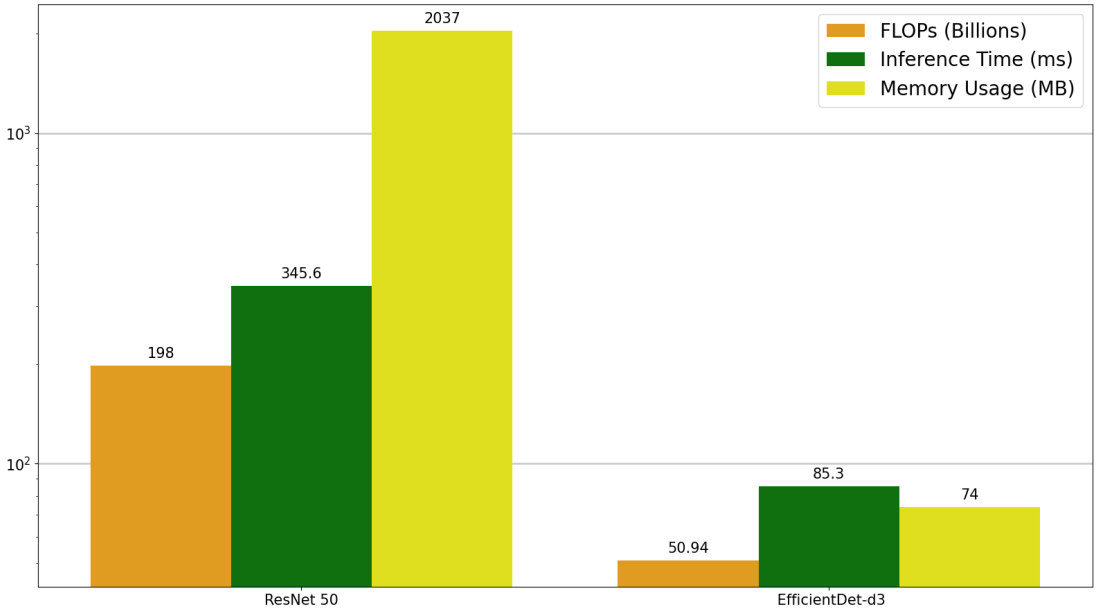


Figure 20: Comparison between CenterMask, with ResNet 50 as backbone and SAM in the mask head and the EfficientDet-d3 with the SAM module.

3.3.3 HTC

In this Section, we will introduce how we have implemented the HTC head to an EfficientDet backbone. As HTC generates the mask in parallel with the bounding boxes in the different stages, we decide to remove the Bounding box and classification heads from EfficientDet. Therefore, we adapt the output of the BiFPN neck to the input of the box prediction branch from the HTC original proposal, removing the FPN and adding an FCN to add more channels to the EfficientDet outputs. Since we use the box and mask heads from HTC, we will need to train to fine-tune both branches and it will lead to more memory and time consumption.

Although it will require more energy to train this algorithm, we assume it will help to improve the results from previous configurations because it will allow us to interconnect the information of the bounding boxes with the mask information. As well as it will raise the training threshold in each stage of the mask head in order to generate better results from previous stages.

This architecture is almost 75% more economical in terms of memory than the original proposal which uses the ResNet 50 as a backbone. In relation to the

doi:10.6342/NTU202203163

number of FLOPs, our implementation with EfficientDet use 32% less than HTC. At the same time that this configuration is faster than the initial design. These information is shown in Figure 21.

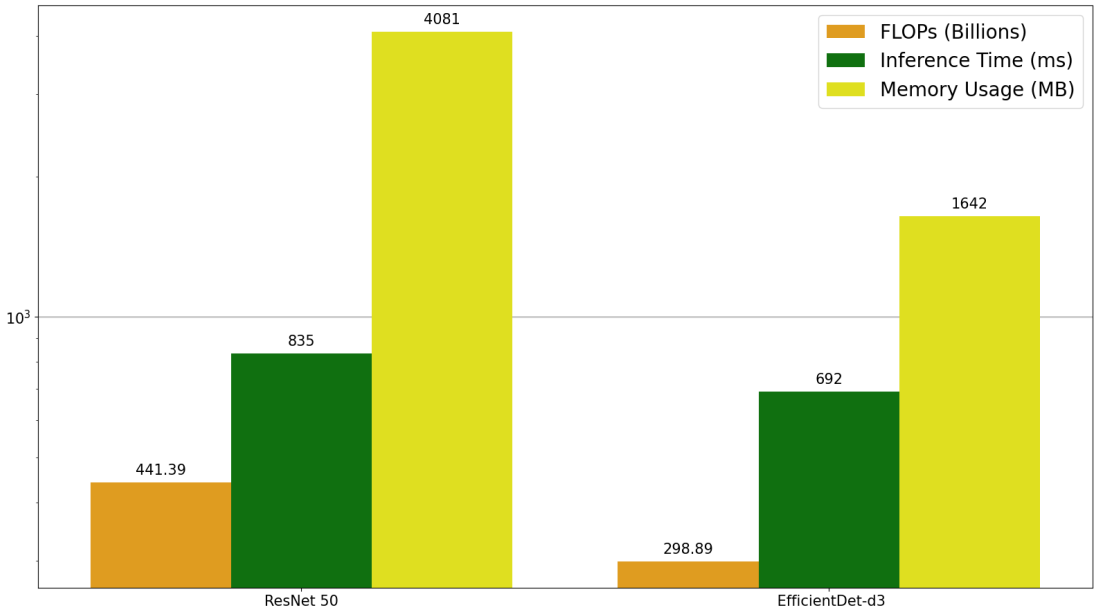


Figure 21: Comparison of the original proposal in the HTC paper [3] and our implementation using EfficientDet-d3 as backbone and neck.

3.3.4 MaskFormer

Following the current direction of the computer vision community of using a transformer architecture, we study the union of the backbone we have chosen and the mask head proposed by [14]. One thing new about this model compared with the previous ones is that it does not rely on the bounding boxes to generate the mask predictions.

As in model 3.3.3, we do not use the box and class prediction branches from the EfficentDet-d3 architecture for this implementation. Doing reference to the preceding paragraph, the mask branch of this model does not base its mask prediction on the RoIs as in the past models. Therefore, we adapt the outputs from $P2$ to $P5$ as inputs of the pixel decoder to apply the MaskFormer branch to the EfficienDet-d3 backbone.

With this architecture, we expect to take advantage of the better performance shown by the transformer architecture in previous papers [14, 22, 4, 11]. However, according to [19] this type of architecture needs more time to train for computer vision tasks. Therefore, we assume it will generate better predictions than the mask branches based on convolutional networks with longer training time.

doi:10.6342/NTU202203163

According to Figure 22, the optimization is not as high as the previous implementation compared with the original model. However, it is still relevant with a decreased of more than 40% in the number of FLOPs and almost a 30% in terms of memory usage.

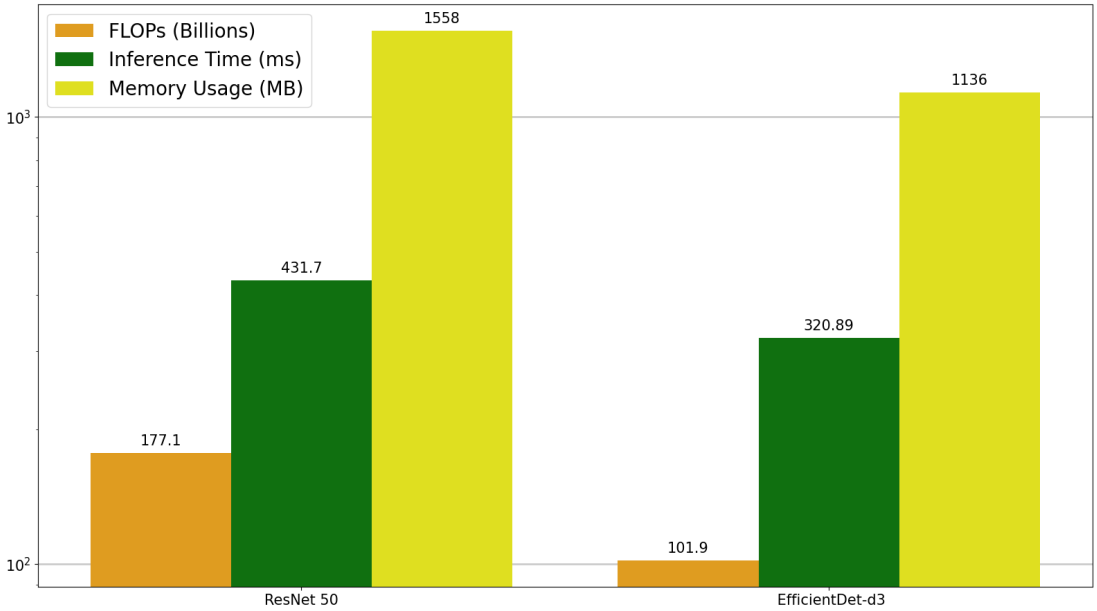


Figure 22: Comparison of the original proposal in the MaskFormer paper [14] and our implementation using EfficientDet-d3 as backbone and neck.

3.3.5 Mask2Former

To continue with the transformer architecture for the mask branch, we decide to analyze the performance of a multi-scale transformer with our backbone. Even though, this head was also proposed by [22] in 2021. Same as in the previous model, we only use the backbone and neck from EfficientDet-d3 and connect the outputs of the BiFPN to the pixel decoder of Mask2Former. For this implementation, We expect to achieve a better performance than the previous model since we can make use of the Multi-scale information produced by EfficientDet. Hence, this architecture will demand more computational capacity than the prior branch. Regarding the performance of this model during training, we can observe in Figure 23 that our implementation is more efficient in relation to computational resource usage. It is important to mention that both models shown are optimal in all terms during the inference time since it will not need to generate the mask in 8 of the 9 transformer decoders.

To sum up, in this chapter we have analyzed several mask heads with a common backbone and neck. We have compared it with the well-known ResNet 50 and FPN architecture in terms of the number of FLOPs, time consumption per image during

doi:10.6342/NTU202203163

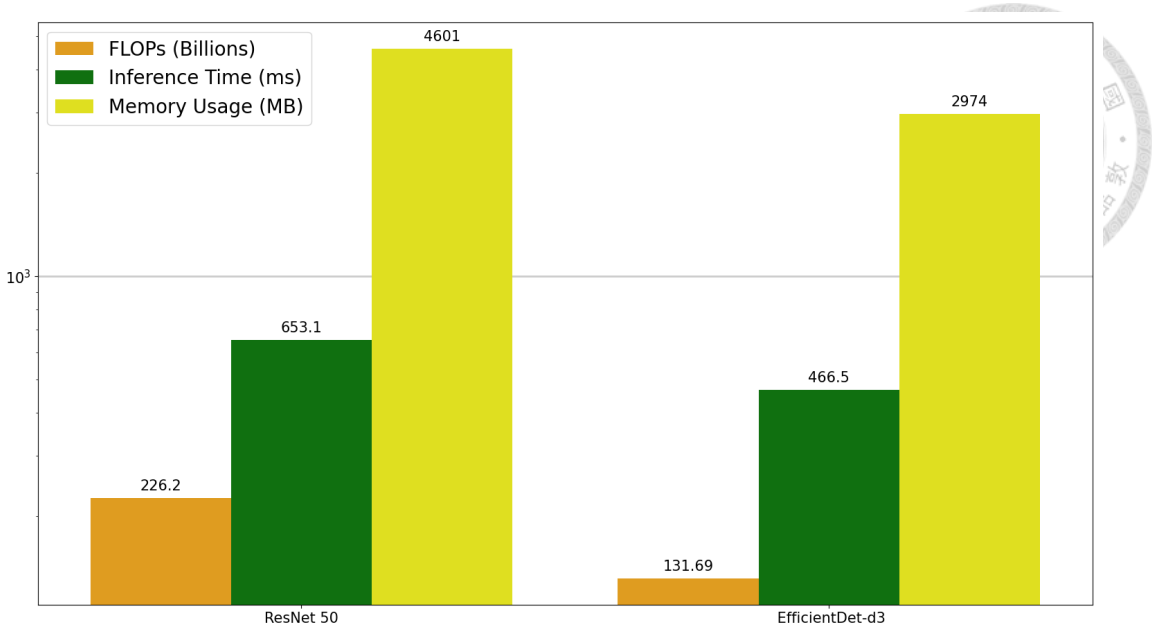


Figure 23: Comparison of the original proposal in the Mask2Former paper [22] and our implementation using EfficientDet-d3 as backbone and neck.

training and the GPU memory used while training an image of 896×896. All the previous results are summarized in Table 5.

Table 5: Comparison of all the models in terms of number of FLOPs, Time, and Memory usage during training with images of 896×896 in an RTX 3060 with 12GB.

Model	Backbone	FLOPs (Billions)	Time (ms)	Memory (MB)
Mask R-CNN	ResNet 50	198	325.5	1960
Mask R-CNN	EfficientDet d3	50.94	83.9	67
Centermask	ResNet 50	198	345.6	2037
Centermask	EfficientDet d3	50.94	85.3	74
HTC	ResNet 50	441.39	835	4081
HTC	EfficientDet d3	298.9	692	1642
MaskFormer	ResNet 50	177.1	431.7	1558
MaskFormer	EfficientDet d3	101.9	320.9	1136
Mask2Former	ResNet 50	226.2	653.1	4601
Mask2Former	EfficientDet d3	131.7	466.5	2974

After our analysis, we found that using EfficientDet as a backbone helps to optimize any model. Since it helps to reduce the number of FLOPs, time consumption and the memory required to train each model. We also can appreciate how models based on convolutional networks which are the top three models in Table 5, require less memory than the transformer-based models. The first two are the smallest models in terms of the number of FLOPs and the reduction of

doi:10.6342/NTU202203163

parameters is more relevant than the rest of the models. Even though, we did not analyze the total number of iterations required to converge each model in this chapter, according to [19] transformer-based model needs a lot more iterations to reach its minimum point. Therefore, the last two models demand more time to be trained than the rest of the models.

Accordingly, with the introduction of this chapter, we can say that all the models proposed above are more optimized than the previous baselines with the ResNet 50. In Chapter 5, we will analyze the performance of each model. In the next chapter, we will introduce the proposal models of this research.

doi:10.6342/NTU202203163

CHAPTER 4

PROPOSED DEEP LEARNING ALGORITHM

This chapter focus in detail on the motivation of this research and the proposed algorithms with an overview of the designs and workflows of algorithms. In this chapter, we will look what is problem scenario of our work. Also, we explain the architectures, components and workflow of methodologies that we select for our thesis.

Based on the models of Chapter 3, we want to use the one with the HTC branch as the baseline network proposal which consists of an adaptation of the EfficientDet and HTC networks to the instance segmentation task. Furthermore, we propose three different modifications to this baseline.

As it is explained in the previous chapter EfficientDet is the most optimal neural network for object detection. It is compounded by EfficientNet as the backbone, BiFPN, and the class and box predictors subnets.

On the other hand, the HTC is the most common branch for instance segmentation. The official model uses ResNet as a backbone, an FPN, a detector head and a mask head. The algorithms proposed are an extension of the EfficientDet by adding a mask head in order to perform an instance segmentation with a lower number of parameters.

4.1 Motivation

We found that most of the achievements accomplished in instance segmentation task in recent years were based on the enhancement of the backbone [4, 5, 6] or the use of data augmentation during training [7, 8, 9]. Both of these approaches carried a raise in the memory and time needed which led to a soar in energy consumption. These facts make it harder to implement these models in real-world scenarios such as autonomous cars. Furthermore, the community is moving towards the transformer architecture, but this structure also requires a high amount of time and resources to achieve good results due to the computational complexity. In order to develop an optimal model which can be easily deployed in real-life applications, we have considered focusing our research on models based on CNN.

HTC [3] architecture is still being used in most of the state-of-the-art instance segmentation models, even though it is a subnetwork based on convolutions. In fact, the 80% of the top 10 use this mask head to generate the masks. This

doi:10.6342/NTU202203163

made us wonder if it is possible to optimize this architecture to obtain better results. In this section, we aim to answer this question by studying three different modifications focus on the implementation of an attention module that we propose in this research.

The main difference between the models is in the mask head of the model. Therefore, they share the same backbone and neck. This is why we will describe first the common components of the proposals. Then, we will introduce our proposals.

Following the implementations of Chapter 3, we use EfficientNet-B3 for our backbone architecture. We can see the configuration of this backbone in Figure 6. Then, we use the outputs from the 4^{th} stage to the 8^{th} which correspond to the levels $P3$ to $P7$ respectively. These feature maps feed 6 sequential layers of BiFPN [27] in order to generate stronger feature maps, similar to the architecture of EfficientDet-d3 mentioned in the previous chapter.

Table 6: EfficientNet-B3 layers [27]. each stage i has a number of layers L_i with an input resolution of $H_i \times W_i$ and an output number of channels of C_i.

Stage i	**Operator** F_i	**Resolution** $H_i \times W_i$	**Channels** C_i	**Layers** L_i
1	conv3 × 3	896 × 896	40	1
2	MBConv1, K3 × 3	448 ×448	24	2
3	MBConv6, K3 × 3	448 ×448	32	3
4	MBConv6, K3 × 3	224 × 224	48	3
5	MBConv6, K3 × 3	112 × 112	96	5
6	MBConv6, K3 × 3	56 × 56	136	5
7	MBConv6, K3 × 3	56 × 56	232	6
8	MBConv6, K3 × 3	28 × 28	384	2
9	Conv1 × 1 & Pooling	28 × 28	1536	1

We can see in Figure 24, how the BiFPN layers are connected to the outputs of the backbone. This neck sets the channels of all the feature maps to 160. Following the original proposal of HTC [3], we use the levels from $P2$ to $P5$ (X_2'' to X_5'' in Fig. 24) to feed the heads in order to generate the bounding boxes and masks. Since the outputs of our neck go from $P3$ to $P7$ we need to get the features of level $P2$ from the 3^{rd} stage of EfficientNet-B3.

Once we get the output feature maps from the last layer of the neck and the backbone, we need to add an additional layer to set the same number of channels to all the feature maps before sending them to the heads. Since the ones coming from the neck have 160 channels and the features of level $P2$ have 32 channels.

doi:10.6342/NTU202203163

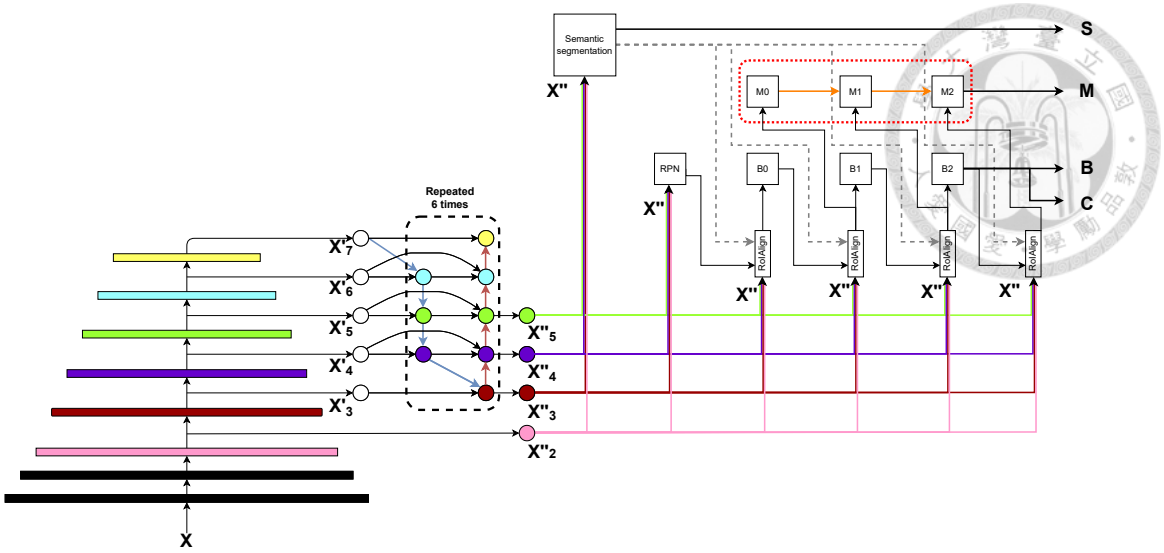


Figure 24: Effi-HTC architecture. RPN is the region proposal network. *Bi* are the different stages of the bounding box head. Inside the red dotted line is the mask branch. The orange arrows are the mask information path between the mask stages.

Therefore, we pass all feature maps to a channel mapper and set all maps to 256 channels.

Afterwards, these feature maps go to the semantic segmentation module which is built based on the FPN [32] architecture. In fact, all the feature maps are re-scale to level $P3$ through 1×1 convolutional layer and then merged by element-wise sum. Later, this feature map goes via four 3×3 convolutional layers before the last 1×1 parallel convolution as shown in Figure 25. The output on the top (S) is the final semantic segmentation prediction of the model, and the one on the bottom (S') is the feature map used for the bounding box branch and the mask branch.

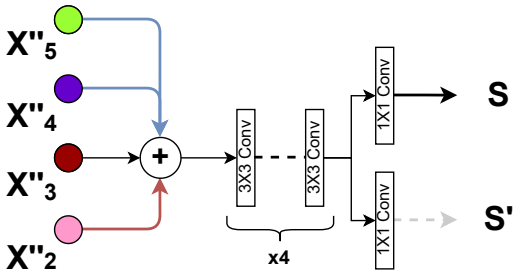


Figure 25: Semantic segmentation module. Blue arrows represent down-sampling and the red arrow refers to an up-sampling function.

On the other hand, the output feature maps of the last BiFPN layers and backbone (from X_2'' to X_5'') are passed to the RPN module. This module use anchors to create the bounding box proposals that will be used as the first RoIs to

doi:10.6342/NTU202203163

feed the first stage of the box head ($B0$). Hence, these proposals are sent to the RoIAlign module to extract the features of each proposal from the neck feature maps (X''), the semantic feature maps (S') and fused them with an element-wise sum. To map the RoI to a feature map level we implement the Equation (2.4).

Then, the fused feature maps feed the first stage of the box head ($B0$) to generate the first bounding box predictions. Each bounding box stage (Bi) is compounded by a shared subnet between the bounding box prediction and the class prediction. The common subnet is composed of two sequential fully connected layers, follow by one subnet branch for each purpose. As shown in Figure 26, each box stage has two inside branches formed by one fully connected layer.

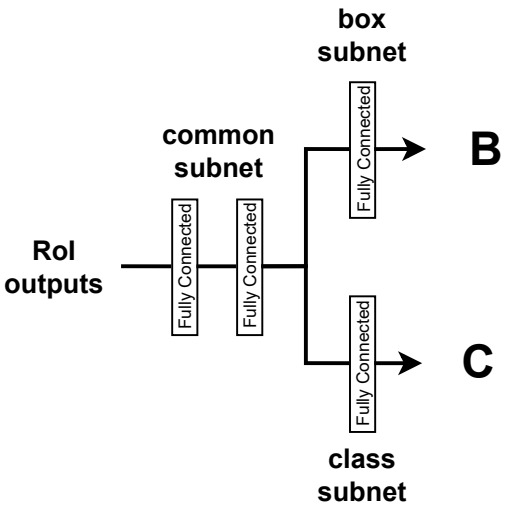


Figure 26: Box stage architecture.

The outputs of the first stage box head are fed to the next RoIAlign module. Then these RoIs are sent to the first stage of the mask head ($M0$) and the second stage of the box head ($B1$). The predictions of this stage are used by the last stage of the bounding box head ($B2$) and the second mask stage($M1$). Therefore, the pipeline is formulated as Algorithm 1.

In Algorithm 1, X'' and S' represent the feature maps of the backbone network and the semantic segmentation branch. $\mathcal{P}(\cdot)$ refers to the RoIAlign Operator. X_t^{box} and X_t^{mask} denote the output feature maps derived for the X'' and the RoIs. B_t and M_t indicate the box and mask t-th stage. b_t and m_t correspond to the bounding box predictions and the mask predictions, respectively. m_{t-1}^{-} represents the intermediate features of M_{t-1}. $\mathcal{F}$ denotes the join function to combine the previous stages features and the current one.

Once we have explained the common part of our proposals, we are going to focus on the main modifications propose to the model of Section 3.3.3. These

doi:10.6342/NTU202203163

Algorithm 1: General Algorithm Pipeline

Input:
$\mathbf{X''}$: feature maps of the backbone network
$\mathbf{S'}$: feature maps of the semantic segmentation branch
Output:
$\mathbf{b_t}$: bounding box predictions of stage *t-th*
$\mathbf{c_t}$: class predictions of stage *t-th*
$\mathbf{m_t}$: mask predictions of stage *t-th*

for $t = 0$ **to** 2 **by** 1 **do**
  $X_t^{box} = \mathcal{P}(X'', b_{t-1}) + \mathcal{P}(S', b_{t-1})$;
  $b_t, c_t = B_t(X_t^{box})$;
  $X_t^{mask} = \mathcal{P}(X'', b_t) + \mathcal{P}(S', b_t)$;
  $m_t = M_t(\mathcal{F}(X_t^{mask}, m_{t-1}^{-}))$; `// This eq will change for each proposal`
end
return b_2, c_2, m_2

variations are located in the mask branch of our model. We can see these modules inside the red box in Figure 24. We named these architectures Hybrid Task Cascade with Attention (HTCA).

4.2 HTCA-IF: HTC with an attention module in the mask information flow

We begin by adding one SAM to each interconnection of the information flow of the HTC mask branch. With this, we want to lead the mask branches to focus more on the information shared between the stages. This is the reason why we named this proposal HTCA-IF. As we show in the previous Chapter 3, this module can enhance the final mask predictions. Therefore, we want to analyze how this attention module would affect the final performance of our model. We can appreciate the architecture of this mask branch in Figure 28a. In our model, the mask head is compounded by three stages. Each stage is fed with the RoI feature maps provided by the RoIAlign module with the information of the bounding box predictions from the box head of the same stage, the semantic features (S') and backbone feature maps, as it is presented in Figure 24. These Feature maps have a size of 14×14. Then, these maps go through the first subnet inside each stage which is composed of four 3×3 convolutional layers.

At this point, we split the path into two ways. One path send the new features maps (m_t^-) to a 2×2 deconvolutional layer which is used to up-sample the feature maps to 28×28. Finally, the final mask prediction of the first stage is generated by a 1×1 convolutional layer.

The other way takes these features to the SAM to help the next stage to focus

doi:10.6342/NTU202203163

on important features and deactivate the rest. we can observe the architecture of this attention module in Figure 27. This part has a maximum pooling layer and average pooling function which outputs feature maps that are concatenated. After the concatenation, there is a 3×3 convolution and this output is normalized by a sigmoid function.

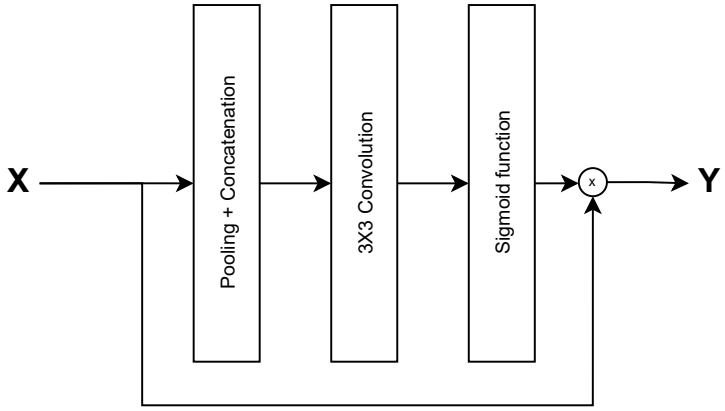


Figure 27: SAM.

The normalized feature maps are combined with the input feature maps of this module via an element-wise multiplication. This way we let this module highlight the important pixels on the feature maps. Following the SAM, there is a 1×1 convolutional layer. These feature maps are fused with the outputs of the next RoIAlign module via an element-wise addition. Hence, in the first proposal, the attention module affects the features used to interconnect the stages. The algorithm pipeline of this model is shown in Algorithm 2.

Algorithm 2: HTCA-IF

Input:
$\mathbf{X''}$: feature maps of the backbone network
$\mathbf{S'}$: feature maps of the semantic segmentation branch

Output:
$\mathbf{b_t}$: bounding box predictions of stage *t-th*
$\mathbf{c_t}$: class predictions of stage *t-th*
$\mathbf{m_t}$: mask predictions of stage *t-th*

for $t = 0$ **to** 2 **by** 1 **do**
  $X_t^{box} = \mathcal{P}(X'', b_{t-1}) + \mathcal{P}(S', b_{t-1})$;
  $b_t, c_t = B_t(X_t^{box})$;
  $X_t^{mask} = \mathcal{P}(X'', b_t) + \mathcal{P}(S', b_t)$;
  $m_t = M_t(\mathcal{F}(X_t^{mask}, \mathcal{SAM}(m_{t-1}^-)))$
end
return b_2, c_2, m_2

The above procedure is repeated in all the following stages of the mask head.

doi:10.6342/NTU202203163

Our objective with this proposal is to increase the performance of the previous model described in Section 3.3.3, by enhancing the quality of the feature maps sent to the next stage to generate the final mask predictions. In the next sections, we will introduce two more proposals.

4.3 HTCA-C: HTC with an attention module in after the convolutional layers

After the first proposal, we found that the attention did not have too much impact on the performance. We think this can be due to the fact that each attention module did not affect directly the final predictions.

This is the reason why we make a second proposal. In this new proposal, we want to add spatial attention in a place where the final predictions can take advantage of this focus function in a more direct way. Considering previous arguments, we decide to insert a SAM Fig. 27 in each stage of the mask head. Thus, the last layers can also exploit higher-quality feature maps from previous layers.

Algorithm 3: HTCA-C

Input:
$\mathbf{X''}$: feature maps of the backbone network
$\mathbf{S'}$: feature maps of the semantic segmentation branch
Output:
$\mathbf{b_t}$: bounding box predictions of stage *t-th*
$\mathbf{c_t}$: class predictions of stage *t-th*
$\mathbf{m_t}$: mask predictions of stage *t-th*

for $t = 0$ **to** 2 **by** 1 **do**
  $X_t^{box} = \mathcal{P}(X'', b_{t-1}) + \mathcal{P}(S', b_{t-1})$;
  $b_t, c_t = B_t(X_t^{box})$;
  $X_t^{mask} = \mathcal{P}(X'', b_t) + \mathcal{P}(S', b_t)$;
  $m_t = M_t(\mathcal{F}(X_t^{mask}, \mathcal{SAM}(m_{t-1}^-)), \mathcal{SAM}(m_t^-))$; `// Attention affect to the signal before the deconvolution and the information flow, see 28b`
end
return b_2, c_2, m_2

As we can see in Figure 28b, there is an attention module in each stage of the mask head. we place this element after the sequential 3×3 convolutional layers right before splitting the paths. In the last stage, we also add a new SAM between the convolutional layers and the deconvolutional layer. Therefore, we led this module to highlight the feature maps which are going to be used to generate the final prediction of each stage. We denote this proposal as HTCA-C.

In this proposal, the output of the attention module goes through the 2×2 deconvolutional layer as well as the information flow path. We believe that this

doi:10.6342/NTU202203163

design can help each stage to generate better masks. Hence, this will lead our model to improve its performance while training.

4.4 HTCA-D: HTC with an attention module before the deconvolutional layer

In this section, we will introduce the last proposal of this thesis. Following the same design idea as the previous proposals, we hope to improve their performance by removing the SAM module from the information flow.

As we can observe in Figure 28c, in this model the attention module is right before the 2×2 deconvolutional layer. Therefore, in this architecture, the attention module is in charge to emphasize the focus only on the final predictions. The signal is used for the generation of the mask predictions. We refer to this model as HTCA-D.

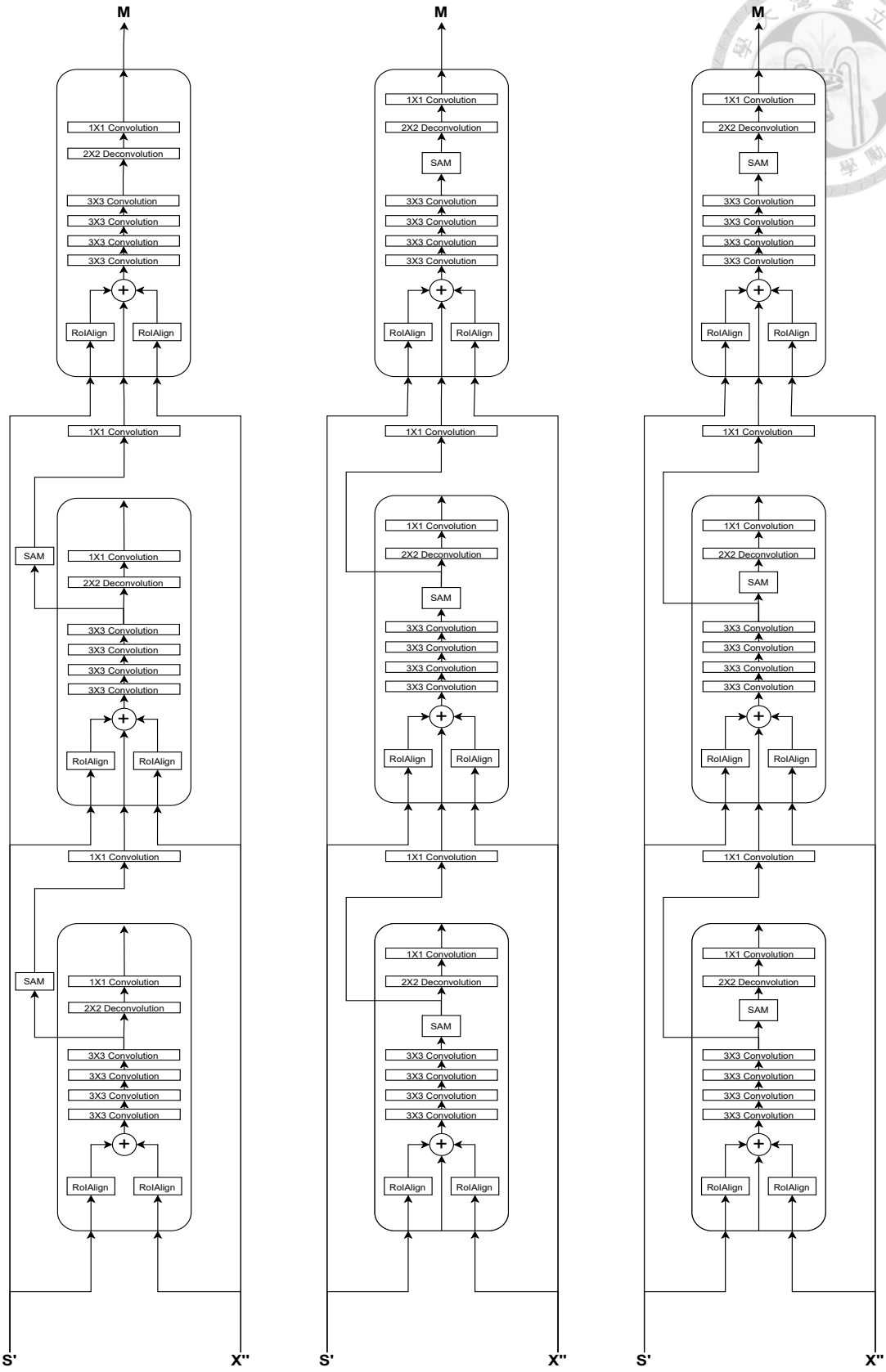
Algorithm 4: HTCA-D

Input:
$\mathbf{X''}$: feature maps of the backbone network
$\mathbf{S'}$: feature maps of the semantic segmentation branch
Output:
$\mathbf{b_t}$: bounding box predictions of stage *t-th*
$\mathbf{c_t}$: class predictions of stage *t-th*
$\mathbf{m_t}$: mask predictions of stage *t-th*

for $t = 0$ **to** 2 **by** 1 **do**
  $X_t^{box} = \mathcal{P}(X'', b_{t-1}) + \mathcal{P}(S', b_{t-1})$;
  $b_t, c_t = B_t(X_t^{box})$;
  $X_t^{mask} = \mathcal{P}(X'', b_t) + \mathcal{P}(S', b_t)$;
  $m_t = M_t(\mathcal{F}(X_t^{mask}, m_{t-1}^-), \mathcal{SAM}(m_t^-))$; `// Attention module only affect to the signal before the deconvolution, see 28c`
end
return b_2, c_2, m_2

As we can see in Algorithm 3, it seems the very similar as Algorithm 4. However, the main difference between them is that in 3 the features maps m_{t-1}^- have gone through the attention module of the previous stage. Meanwhile in Algorithm 4 does not apply attention to the feature maps of the information flow m_{t-1}^-.

doi:10.6342/NTU202203163



(a) HTCA-IF (b) HTCA-C (c) HTCA-D

Figure 28: Models Proposed. (a) HTCA-IF. The SAM in the information flow between stages. (b) HTCA-C. The SAM at the exit of the last convolution. (c) HTCA-D. The SAM is located before the deconvolutional layer.

doi:10.6342/NTU202203163

CHAPTER 5

PERFORMANCE EVALUATION

In this chapter, the evaluation of the proposed methods is presented. Firstly, the simulation settings are introduced with an explanation of the dataset used and its metrics. Then, we explain the training hyper-parameters. Finally, we evaluate the performance of the proposed algorithms in comparison with other instance segmentation models.

5.1 Implementations Details

In our research, we perform a comparison of our models to the state-of-the-art on COCO dataset [37]. We used the standard COCO metrics including AP(average over IoU thresholds 0.5-0.95), AP_{50}, AP_{75}, and AP_s, AP_M, AP_L. Where S , M and L mean Small, Medium and Large, and define the different scales. In this chapter when we talk about the AP, we refer to the AP which is evaluating the mask IoU. Otherwise, we will refer to the evaluation of the bounding boxes IoU as AP^{bb}.

As in previous works [1, 2, 3, 14, 22], we join the 80k train images with a 35K subset of validation images to train our model and perform the test on the remaining 5k validation images, also known as minival.

In this section, we will discuss the different configurations we have used to train each of the models explained in previous chapters.

The first two models, it is the Mask R-CNN and Centermask with efficientDet-d3 as the backbone. For these models, we have followed the same hyper-parameters as in the official paper [1]. First, we use the whole architecture of EfficientDet-d3 with the official pre-trained weights. We have trained these models with the multi-task loss (5.1) on each RoI.

$$L = L_{mask}, \tag{5.1}$$

where L_{mask} is the mask loss, we use a binary cross-entropy loss on the RoIs that are associated with the ground truth.

For the training schedule, we have followed one schedule, x1 which is a convention proposed by Facebook AI Research Group. This schedule requires training the model for 12 epochs and decreasing the learning rate by 10 after the 8^{th} and 11^{th} epochs. An epoch in this field means that the entire dataset is passed forward

doi:10.6342/NTU202203163

and backward through the neural network only once. Furthermore, they proposed two more schedules x2 and x3 which require 24 and 37 epochs, respectively. We train these models on 1 GPU with a batch size of 16 images. We use a weight decay of 0.0001 and momentum of 0.9, with a starting learning rate of 0.02.

The models that use the HTC mask branch have a similar training implementation as the ones above. However, we only use the pre-trained backbone and neck without the box head. Moreover, we use a different loss function to train these models. Since in these models we have to train the box head and the mask head, we set a different weight for each stage. The overall loss function is shown in Equation (5.2) as follows:

$$\begin{aligned} L &= \textstyle\sum_{t=0}^{T} \alpha_t (L_{bbox}^t + L_{mask}^t) + \beta L_{seg}, \\ L_{bbox}^t &= L_{cls}^t + L_{reg}^t, \end{aligned} \tag{5.2}$$

where L_{bbox}^t refers bounding box stages losses. L_{cls}^t and L_{reg}^t denote the classification and regression losses, respectively. For α and β, we set the same parameters as the original paper [3] which are $[1, 0.5, 0.25]$ for α and $\beta = 1$. L_{mask}^t represents the mask binary cross-entropy loss of each stage.

Moreover, we trained these models following the x1 schedule with an initial learning rate of 0.02. Since all the models with this head required a high amount of memory, we train them with 1 image per batch.

To end this section, we describe the configuration details of the models with a transformer-based architecture which are MaskFormer and Mask2Former. We also use the pre-trained backbone and neck from EfficientDet-d3. For these models, we follow a similar configuration as in Mask2Former [22]. As mentioned before these models need more iterations to converge. Therefore, we train these models for 50 epochs and decrease the learning rate by 10 after 44 and 48 epochs.

Furthermore, we also change the loss function to train these models to make a fair comparison with the official models. The loss function of these models is applied to all the transformer decoder outputs. we use a binary cross-entropy loss as in the rest of the models plus a dice loss. The overall loss function is shown in Equation (5.3)as follows:

$$\begin{aligned} L &= L_{mask} + \lambda_{cls} L_{cls}, \\ L_{mask} &= \lambda_{ce} L_{ce} + \lambda_{dice} L_{dice}, \end{aligned} \tag{5.3}$$

where λ_{ce} and λ_{dice} denote the weights of the different mask losses and they are set to 5 ($\lambda_{ce} = \lambda_{dice} = 5$). λ_{cls} refers to the classification weight which is set to 2 ($\lambda_{cls} = 2$).

As well as previous models, these architectures demand a lot of memory while

doi:10.6342/NTU202203163

training them. Hence, we set only 1 image per batch with an initial learning rate of 0.0001. These models require so much memory that we can not train the last model explained in Chapter 3, the one with the Mask2Former branch, in an RTX 3060 GPU with 12 GB of memory. We move some modules of this branch to the CPU during the training, but it would take almost a year to train it for 50 epochs. Therefore, we do not analyze its performance in this work.

5.2 Main Results

In this section, we present all the results obtained in this work. We start introducing the results of the Mask R-CNN and the CenterMask together. Then, we show the performance of the HTC models and the proposals in Chapter 4. Finally, we discuss the results obtained with the MaskFormer models.

First of all, we compare the Mask R-CNN and the CenterMask with their official baselines on the COCO validation dataset in Table 7. For the Mask R-CNN, we can see on the table that there are two rows with the same model but different values. This is because the values of the first Mask R-CNN row are from the initial result from the official paper, and the row below is the latest performance from the official GitHub repository. Moreover, the results of the CenterMask are from the official GitHub repository. In addition, the models with EfficientDet-d3 outperform all the baselines with ResNet-50 as the backbone. In the instance segmentation, our Mask R-CNN configuration improves the latest result by 0.5 AP points with a third of the training iterations. For the CenterMask model, we obtain +1.4 AP improvement over the official performance with half of the training schedule. Moreover, we like to highlight the reduction in the number of parameters by 66% and the number of FLOPs by a 75%.

Table 7: Mask R-CNN and CenterMask results. The first Mask R-CNN results are from the official publication [1] and the second row is from the official GitHub repository [38]. The CenterMask results are from the official GitHub repository [39].

Model	Backbone	AP^{bb}	AP	AP_S	AP_M	AP_L	lr sched	Params (M)	FLOPs (M)
Mask R-CNN	R50	37.3	33.7	14.4	36.6	50.8	x1	44.3	198.0
Mask R-CNN	R50	41.0	37.2	18.6	39.5	53.3	x3	44.3	198.0
Mask R-CNN	Effi-d3	46.0	37.7	18.9	41.7	55.0	x1	14.4	50.9
CenterMask	R50	41.2	36.4	17.3	39.5	52.7	x2	44.3	198.0
CenterMask	Effi-d3	46.0	37.8	18.9	41.9	55.2	x1	14.4	50.94

Table 8 shows the performances of the HTC models and the proposals on the COCO validation dataset. In this case, we have trained the HTC model ourselves in order to make a fair analysis. This is why the HTC with the ResNet 50 does

doi:10.6342/NTU202203163

not match the official result. Even though, we have used the official source code to train it. This difference is due to the fact that the official result is trained it with data augmentation and they also trained the backbone during the training while we used the ResNet 50 pretrained model. In Table 8, we can observe how the HTC with EfficientDet as backbone enhances the model mask precision by +0.5 points with 30% less of FLOPs.

Since the HTC with the EfficientDet-d3 backbone has a better performance we decide to use this architecture to analyze the performance of our proposals. We also made this decision based on the time required to train each model in our GPU. The model with ResNet 50 takes 11 days to be trained with an x1 schedule around double the time needed by the model with EfficientDet.

Now, we proceed to explain our proposal and compare them with the HTC baseline with the EfficientDet backbone. We can observe in Table 8 that HTCA-IF is the one with the worse overall performance. The second proposal, HTCA-C with the attention component affecting each stage individually obtained the best improvement with +1.2 AP points over our baseline model 3.3.3. In the last proposal, HTCA-D achieved a better performance than the second model by +0.1 AP point. Moreover, it enhances the baseline model by +1.3 AP points. Furthermore, we can appreciate how our proposals also help to improve the performance of the object detector. In addition, we see how the models with EfficientDet as backbone perform better in larger objects than smaller ones compared with the official architecture.

Table 8: HTC and HTCA results on COCO val dataset.

Model	Backbone	AP^{bb}	AP	AP_S	AP_M	AP_L	lr sched	Params (M)	FLOPs (G)	Training (days)
HTC	R50	37.1	33.6	15.7	35.9	51.0	x1	80.0	441.4	11
HTC	Effi-d3	40.3	34.1	11.1	38.7	55.4	x1	66.6	281.0	5
HTCA-IF	Effi-d3	40.1	33.9	10.8	38.7	54.8	x1	66.6	281.0	5
HTCA-C	Effi-d3	41.7	35.3	12.9	39.6	55.5	x1	66.6	281.0	5
HTCA-D	Effi-d3	41.6	35.4	13.0	39.8	55.8	x1	66.6	281.0	5

In our analysis, we found that the bad performance of the HTCA-IF model is due to the fact that the attention is misleading in the following mask stages. In Fig. 29 we can see that this happens because when we applied spatial attention to highlight the common features between stages this caused a deactivation of some parts of the object. Our baseline with EfficientDet as the backbone does not highlight any part of the shape. In the HTCA-IF, the feature map seems similar to our baseline but we can see how the right arm is melted with the background. This difference between both models is caused because the convolutional layers after the attention module reduce the emphasis of the attention by the element-wise sum with the RoIs.

doi:10.6342/NTU202203163

On the other hand, there is a clear difference between both HTC models. Using EfficientDet as the backbone helps the model to extract higher quality feature maps that help the algorithm to remark the silhouette of the person. Even though, the Model with the ResNet 50 highlight more the back of the person in Figure 29. It does not perform well to make a sharp differentiation of the shape of the object compared with the HTC model with EfficientDet. This caused the poor results of the HTC initial proposal in spite of having a greater number of parameters

Moreover, the features in the last stage of the HTCA-C highlight more than the baseline of the man's back which led to enhancing the final masks. To conclude this discussion, the feature maps generated by HTCA-D are the ones that emphasize the back of the man the most. This helps the last layers to make a more accurate prediction than the other models. To place the attention module before the deconvolutional layer also helps to point out the background around the main object similar to the original proposal which has a better performance.

In the last table, we compare the performance of the transformer-based models. As mentioned above we only analyze the MaskFormer (Sect. 3.3.4) due to the hardware limitations. In Table 9, It can observed how the model with EfficientDet-d3 as backbone outperforms the official result by $+1$ AP point. Moreover, it also shows how the model with the ResNet 50 backbone does have a better performance in AP_S. This also happened with the algorithm in Table 8, in which the models with EfficientDet as backbone achieve a higher result with larger objects.

In addition, it should highlight the fact that our model was able to achieve a better performance requiring $6\times$ fewer training iterations. Besides, it also reduced the number of parameters by more than the 25%, and the number of FLOPs decreased by almost half of the model with ResNet Backbone.

Table 9: MaskFormer results. The results of the MaskFormer with ResNet 50 as backbone were obtained from the official paper [22].

Model	Backbone	AP	AP_S	AP_M	AP_L	epochs	Params (M)	FLOPs (G)
MaskFormer	R50	34.0	16.4	37.8	54.2	300	45.0	177.1
MaskFormer	Effi-d3	35.0	11.8	38.6	58.6	50	32.7	101.9

As we can see in Tables 8 and 9, the models which Using EfficientDet as the backbone tend to have a lower performance in smaller objects than the models with the ResNet-50. This is due to the fact that the models with the ResNet use the feature maps generated by the last layers of the backbone ($P2$ to $P5$), meanwhile, the models with EfficientDet-d3 are using the same levels when they generate feature maps of two more levels($P6$ and $P7$). Since the last two levels

doi:10.6342/NTU202203163

are the ones with smaller scales, they tend to highlight the features of the small objects. However, we leave this analysis for future research.

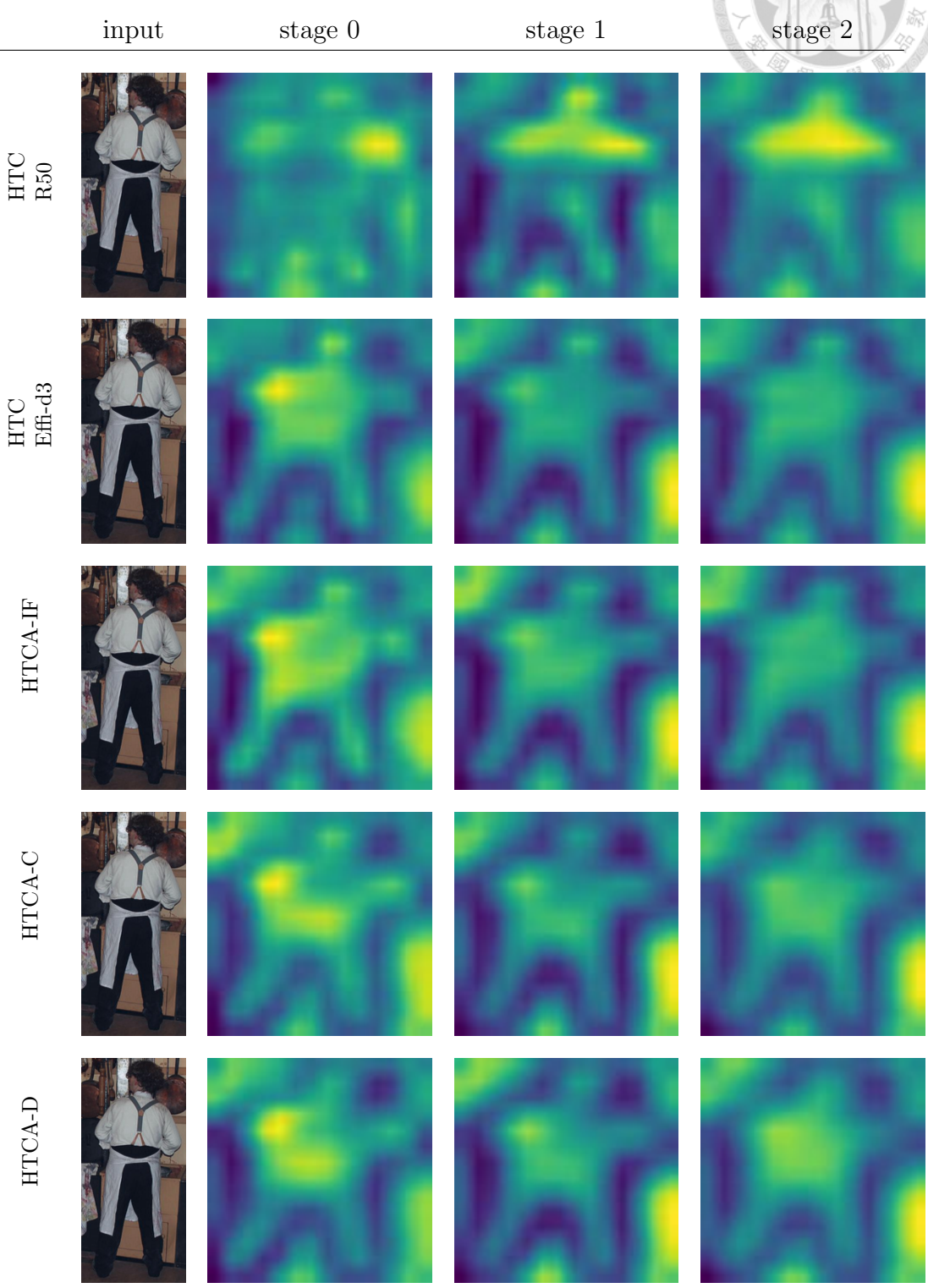


Figure 29: Average feature map before the deconvolutional layer in each stage.

doi:10.6342/NTU202203163

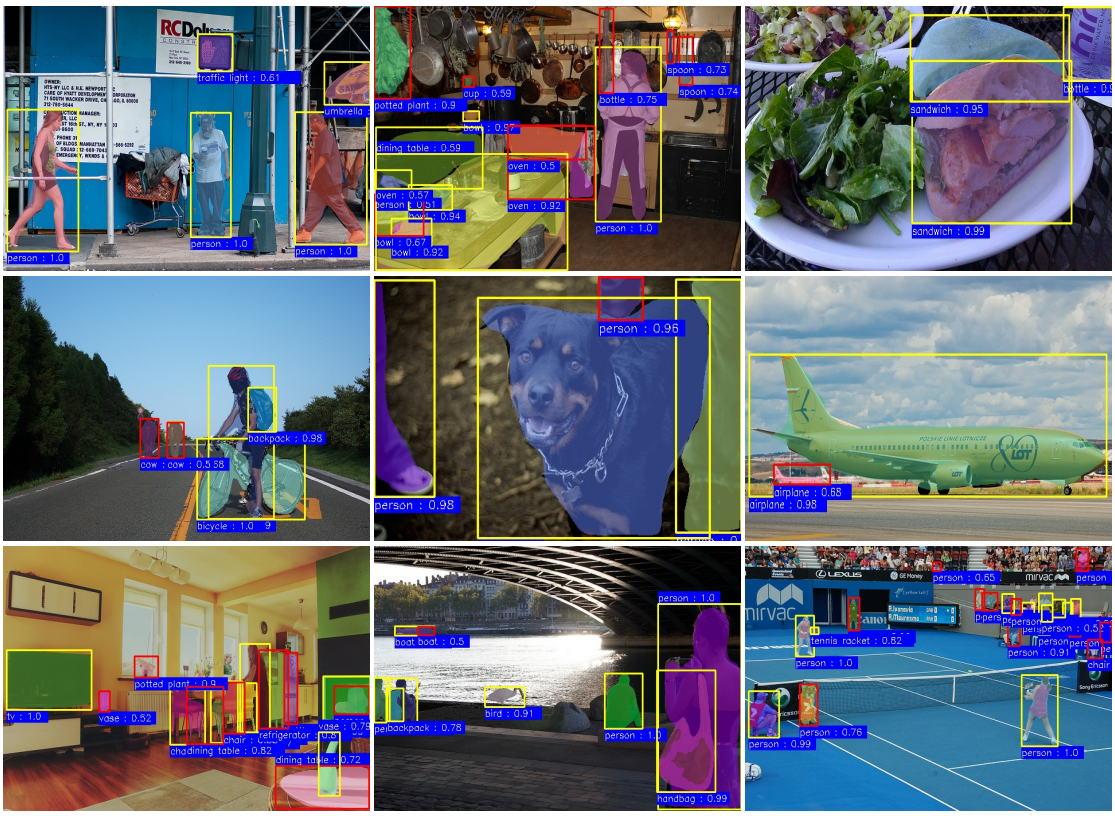


Figure 30: Examples of instance segmentation results on COCO dataset.

doi:10.6342/NTU202203163

CHAPTER 6

CONCLUSION AND FUTURE WORK

We have investigated the design of multiple instance segmentation algorithms in order to get a higher quality of mask predictions and optimize the baseline models. The impact of a more optimal backbone and several mask head architectures are investigated in this work.

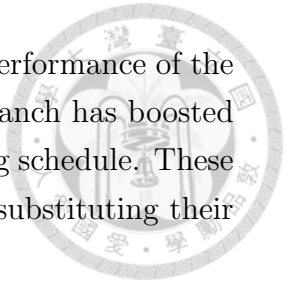
We found that new backbones and data augmentations procedures have helped to improve instance segmentation models. It is also true that the new transformer architecture has contributed to these improvements too. However, we think that these approaches are not suitable for real-world applications since they are inefficient. Moreover, we discovered that most of the instance segmentation state-of-the-art algorithms still relaying in the HTC branch which was proposed back in 2019. And new branch proposals were based on transformer architecture making them more complex.

Our analysis shows how these models are getting more expensive in terms of computational complexity and hardware resources need it. This analysis has made us wonder if it would be possible to optimize some of these recent models by making some changes in their structures.

Furthermore, we noticed that the CNN-based models were getting unpopular in the computer science community since the release of the transformer architecture. However, transformer-based algorithms are heavier than CNN-based models and they also required a longer time to converge.

After the analysis of the related work, we found that the HTC model was inefficient and was missing focus in the mask generation branch. This is why we have analyzed three new architectures for instance segmentation task to enhance the mask head of the well-known HTC. The results show that adding attention modules in all the different stages helps the baseline to get a higher performance. We also have analyzed the use of the EfficientDet architecture as backbone and neck with multiple mask heads. In this research, we showed that EfficientDet architecture can help to improve all of the baselines by reducing the computation complexity and the memory required, which led to a decrease in the training time required. In the Mask R-CNN, the improvement of the performance was +0.5 AP points with only a 25% of FLOPs and without data augmentation which is a very impressive achievement. For the CenterMask structure, our configuration beat the initial proposal with the ResNet 50 by +1.4 AP points. We also found that

doi:10.6342/NTU202203163



the EfficientDet backbone can also help to improve the overall performance of the transformer-based mask heads. In our case, the MaskFormer branch has boosted its official performance by +1 AP in a sixth of the official training schedule. These results show that all the baselines could be easily improved by substituting their backbones.

Moreover, we have investigated the impact of placing a SAM in the different connections of the mask stages. We have found that HTCA-IF which adds extra attention to mask information flow does not help to improve the performance. However, placing the attention module in each stage before predicting the masks as in HTCA-C and HTCA-D, can benefit the overall performance of the model as well as boosts the bounding box predictor. In our final proposal, HTCA-D increased the overall performance of the model by +1.3 AP compared with the baseline. This enhancement is obtained by applying attention right before the deconvolutional layer of each stage.

In addition, we would like to propose some future lines to continue this work. As stated before, the Efficientdet-d3 performs well compared to the ResNet 50. It would be interesting to get a deeper comparison with the larger configuration ResNet-101 or 152 with the EfficientDet-d7. It would be interesting to use all the outputs(X''_{3-7}) from the BiFPN, or even train a BiFPN which also has the features maps of level $P2$ as inputs. Another suggestion for future research would be to train our proposal with a ResNet backbone to study its performance with the original architecture. We think it would be interesting to study the performance of the Mask2Former branch with EfficientDet (Sect. 3.3.5).

Furthermore, we found it would be very interesting to make a deeper analysis of the energy consumption in terms of the number of FLOPs, the time needed for the training and the GPU used. Another future line would be to train the backbone and the neck with the heads instead of using a pre-trained model which was not possible for our lack of resources. Finally, we believe that the models proposed in this thesis could be further enhanced with the help of data augmentation techniques.

doi:10.6342/NTU202203163

REFERENCES

[1] Kaiming He, Georgia Gkioxari, Piotr Dollr, and Ross Girshick. Mask r-cnn, 2017.

[2] Youngwan Lee and Jongyoul Park. Centermask : Real-time anchor-free instance segmentation, 2020.

[3] Kai Chen, Jiangmiao Pang, Jiaqi Wang, Yu Xiong, Xiaoxiao Li, Shuyang Sun, Wansen Feng, Ziwei Liu, Jianping Shi, Wanli Ouyang, Chen Change Loy, and Dahua Lin. Hybrid task cascade for instance segmentation, 2019.

[4] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows, 2021.

[5] Tingting Liang, Xiaojie Chu, Yudong Liu, Yongtao Wang, Zhi Tang, Wei Chu, Jingdong Chen, and Haibin Ling. Cbnetv2: A composite backbone network architecture for object detection, 2021.

[6] Ze Liu, Han Hu, Yutong Lin, Zhuliang Yao, Zhenda Xie, Yixuan Wei, Jia Ning, Yue Cao, Zheng Zhang, Li Dong, Furu Wei, and Baining Guo. Swin transformer v2: Scaling up capacity and resolution. 2021.

[7] Mengde Xu, Zheng Zhang, Han Hu, Jianfeng Wang, Lijuan Wang, Fangyun Wei, Xiang Bai, and Zicheng Liu. End-to-end semi-supervised object detection with soft teacher, 2021.

[8] Golnaz Ghiasi, Yin Cui, Aravind Srinivas, Rui Qian, Tsung-Yi Lin, Ekin D. Cubuk, Quoc V. Le, and Barret Zoph. Simple copy-paste is a strong data augmentation method for instance segmentation, 2020.

[9] Hao-Shu Fang, Jianhua Sun, Runzhong Wang, Minghao Gou, Yong-Lu Li, and Cewu Lu. Instaboost: Boosting instance segmentation via probability map guided copy-pasting, 2019.

[10] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017.

doi:10.6342/NTU202203163

[11] Jianwei Yang, Chunyuan Li, Pengchuan Zhang, Xiyang Dai, Bin Xiao, Lu Yuan, and Jianfeng Gao. Focal self-attention for local-global interactions in vision transformers, 2021.

[12] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The cityscapes dataset for semantic urban scene understanding. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[13] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation, 2014.

[14] Bowen Cheng, Alexander G. Schwing, and Alexander Kirillov. Per-pixel classification is not all you need for semantic segmentation, 2021.

[15] M. Everingham, S. M. A. Eslami, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The pascal visual object classes challenge: A retrospective. *International Journal of Computer Vision*, 111(1):98–136, jan 2015.

[16] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.

[17] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection, 2017.

[18] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017.

[19] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale, 2020.

[20] Shu Liu, Lu Qi, Haifang Qin, Jianping Shi, and Jiaya Jia. Path aggregation network for instance segmentation, 2018.

[21] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks, 2015.

[22] Bowen Cheng, Ishan Misra, Alexander G. Schwing, Alexander Kirillov, and Rohit Girdhar. Masked-attention mask transformer for universal image segmentation, 2021.

doi:10.6342/NTU202203163

[23] Feng Li, Hao Zhang, Huaizhe xu, Shilong Liu, Lei Zhang, Lionel M. Ni, and Heung-Yeung Shum. Mask dino: Towards a unified transformer-based framework for object detection and segmentation, 2022.

[24] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

[25] Xianzhi Du, Tsung-Yi Lin, Pengchong Jin, Golnaz Ghiasi, Mingxing Tan, Yin Cui, Quoc V. Le, and Xiaodan Song. Spinenet: Learning scale-permuted backbone for recognition and localization, 2020.

[26] Mingxing Tan and Quoc V. Le. Efficientnet: Rethinking model scaling for convolutional neural networks, 2020.

[27] Mingxing Tan, Ruoming Pang, and Quoc V. Le. Efficientdet: Scalable and efficient object detection, 2019.

[28] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew Howard, and Quoc V. Le. Mnasnet: Platform-aware neural architecture search for mobile, 2019.

[29] Barret Zoph and Quoc V. Le. Neural architecture search with reinforcement learning, 2017.

[30] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. 2018.

[31] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

[32] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection, 2017.

[33] Jifeng Dai, Kaiming He, and Jian Sun. Instance-aware semantic segmentation via multi-task network cascades, 2015.

[34] Ross Girshick. Fast r-cnn, 2015.

[35] Xizhou Zhu, Weijie Su, Lewei Lu, Bin Li, Xiaogang Wang, and Jifeng Dai. Deformable detr: Deformable transformers for end-to-end object detection, 2020.

[36] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection, 2017.

doi:10.6342/NTU202203163

[37] Tsung-Yi Lin, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence Zitnick, and Piotr Dollár. Microsoft coco: Common objects in context, 2015.

[38] FAIR. Detectron 2 repository. [Last Access] 2022-07-29.

[39] Youngwan Lee. Centermask repository. [Last Access] 2022-07-29.

doi:10.6342/NTU202203163