Small-Object Detection in Remote Sensing Images and Video

by

Stamatios Orfanos

Submitted
in partial fulfilment of the requirements for the degree of

Master of Artificial Intelligence
at the
UNIVERSITY OF PIRAEUS
July 2024

Author..... **Stamatios Orfanos**
II-MSc "Artificial Intelligence"
, 2024

Certified by..... **Ilias Maglogiannis**
Professor
Thesis Supervisor

Certified by..... **Theodoros Giannakopoylos**
Researcher
Member of Examination Committee

Certified by..... **Michael Filippakis**
Professor
Member of Examination Committee

Small-Object Detection in Remote Sensing Images and Video

By

Stamatios Orfanos

Submitted to the II-MSc "Artificial Intelligence" on XX XX, 2024, in
partial fulfilment of the
requirements for the MSc degree

Abstract

Object detection in remote sensing images has been a challenging problem for the computer vision research community because the objects in such images have very few pixels. There have been improvements in the mean Average Precision (mAP) of the models using different architectures. Most of the detection models are becoming more complex and bigger, which can cause a problem usually when a detection model is intended for use in a satellite or an Unmanned Aerial Vehicle, since their computation resources are limited. The thesis introduces a novel approach that has achieved a comparable in computational complexity, to the lightest models in the field. The model was able to achieve an substantial improvement on the Unmanned Aerial Vehicle Small Object Detection (UAV-SOD) and the Visual Drone (visdrone) datasets in the order of x% and y% accordingly, while maintaining an almost identical performance on the last and most complex dataset the Microsoft Common Object in COntext (MS COCO) at z%.

The results demonstrate the effectiveness of the proposed method, providing useful intel in multi-task learning and achieving better than most computational efficiency with detection performance for the utilized datasets.

Thesis Supervisor: Ilias Maglogiannis
Title: Professor

Contents

List of Figures

List of Tables

1 Introduction

Remote sensing imaging is a process used to gather information about objects or areas from a distance, typically using aircraft or satellites. This process is essential in various fields due to its ability to detect and monitor the physical characteristics of an area by measuring its reflected and emitted radiation.

Remote sensing imaging starts with data acquisition through sensors, that are mounted on platforms like satellites or aircraft, capture electromagnetic radiation and can range from simple cameras to complex radar systems. After capturing this data, it is transmitted to ground stations for processing. The processing stage often involves correcting any image distortions, enhancing details, and converting the raw data into usable formats.

Remote sensing imaging has applications across a broad spectrum of fields. Starting with environmental monitoring is one of the primary uses, enabling the observation and analysis of environmental changes like deforestation and the health of aquatic ecosystems. In agriculture, it helps monitor crop health and soil conditions, aiding in the efficient management of resources. The technique is also crucial in disaster management, where it is used to assess damage from natural disasters and plan effective responses. Urban planning benefits from remote sensing by providing data for the development and monitoring of infrastructure and urban growth. In the military and intelligence sectors, remote sensing is key for surveillance and reconnaissance, providing critical information for national security.

Provided the numerous applications of remote sensing images, the computer vision research community is continually pushing to develop object detection models that can effectively parse and interpret the vast amount of data captured by remote sensors. In remote sensing images the objects are small fraction of the pixels of the image, qualifying this process as Small Object Detection.

Even though impressive results have been achieved on large and medium objects in large-scale detection benchmarks, the performance on small or tiny objects is far from satisfactory. Compared with large and medium objects, the small objects are more difficult to detect accurately, because of four main difficulties. Firstly, small objects have low resolution and insufficient features. Secondly, the span of object-scale is large and multiple scales coexist. Thirdly, the examples of small objects are scarce and lastly the categories for small objects are imbalanced for the majority of datasets.The concept of small or tiny objects seeks to elucidate the scale of these objects or the proportion of pixels they occupy within the entire image. There are two main ways to define small objects.

The first way is the use relative size. According to the definition of Society of Photo-Optical Instrumentation Engineers [SPIE], if the object size is less than 0.12% of the original image, it is regarded as a small object. Following the same principle Krishna and Jawahar [1] showed that an object is considered small if it occupies only a tiny portion of the image, which is less than 1% of the image area. Namely, the bounding box of a small object should cover less than 1% of the original image. The second way of defining a small object by using the absolute size, where a small object has size less than 32x32 pixels defined in MS-COCO dataset or 16x16 pixels defined in USC-GRAD- STDdb [2].

There have been some improvements of the models using different techniques, but all these improvements come at a increased complexity and size of the model. This complexity can be prohibitive for applications in a satellite or an Unmanned Aerial Vehicle since their computation capabilities are limited. Driven by the need for more precise object detection models, this thesis proposes a novel methodology to reduce the computation cost of the detection model for utilization in such cases.

This thesis aims to explore the combination of two successful models from two different approaches in object detection, while maintaining a smaller model size. The evaluation process utilizes datasets that have been parts of employed in the original research papers of these models. Furthermore, this thesis extends its scope to the field of Remote Sensing Images (RSI). Both the original and modified versions of the models will be evaluated using a common RSI dataset. This will facilitate a comprehensive comparison of all model results within a consistent dataset, thereby providing valuable insights into their performance in real-world scenarios.

The remainder of the thesis is organized as follows:

Chapter 2 contains related work surrounding the scope of the thesis. It starts with an explanation of the architecture of Recurrent Convolutional Neural Networks (R-CNNs) alongside the Feature Pyramid Networks and analyzes the distinct role and functionality of each component. It follows with the explanation of the architecture of the Vision Transformers that were used as a basis for the detector of our model. It continues by explaining the difficulties of detecting small objects in remote sensing images.

Chapter 3 introduces the architecture of the suggested model and is presented, providing an extensive description of its design and functionality. It also highlights the significant publications and research that have been a major help in advancing and improving the model's design.

Chapter 4 offers an in-depth overview of the datasets used to evaluate the proposed model. It states the specific parameters used throughout the training phase to ensure a thorough knowledge of the model's learning process. The experimental findings are presented at the end of the chapter, providing a concrete indicator of the model's effectiveness.

Chapter 5 analyzes the experimental findings and discuss the implications of the differences that were observed across different models and datasets. This aims to unravel the underlying implications of these differences, thereby enhancing the understanding of the models' performance across different datasets.

Chapter 6 provides an overview of the future work which aims at investigating performance differences, enhancing the model's performance and testing the method's generalizability across various datasets and domains.

2 Related Work

This chapter highlights the theories and recent advancements in the fields of remote sensing and computer vision, particularly in small object detection. By examining previous research that addresses those challenges, this section not only underscores the technological progress achieved but also identifies the gaps that the current model aims to bridge. In the field of computer vision, Convolutional Neural Networks (CNNs) served as the initial models for image analysis, primarily focused on image classification where the entire image is labeled as a single object category. While CNNs had great performance in these tasks, their application to object detection in complex images revealed significant limitations.

Object detection models can be broadly categorized by annotation method into anchor-based and anchor-free approaches, and by detection method into dense prediction (single-stage) and sparse prediction (multi-stage) techniques. This categorization helps in understanding the diverse strategies employed in detecting objects across different scenarios.

This chapter begins with the foundational R-CNN model, which introduced the use of convolutional networks for robust object detection. The following sections dive into Fast R-CNN and Faster R-CNN, which iteratively refined the integration of region proposal mechanisms with deep learning, significantly enhancing detection efficiency and speed. The exploration continues with Mask R-CNN and Feature Pyramid Networks, which extended capabilities to instance segmentation and improved feature representation at multiple scales, respectively.

Single-stage detection models, or dense prediction models, such as YOLO (You Only Look Once) and SSD (Single Shot MultiBox Detector), streamline the detection process by eliminating the need for a separate region proposal step. These models directly predict object classes and bounding box coordinates from full images in one evaluation, optimizing for speed and efficiency.

The latter sections of the chapter explore cutting-edge developments like the Vision Transformer and Masked-Attention Mask Transformer, which incorporate transformer architectures to push the boundaries of object detection and segmentation.

2.1 Multi-stage Object Detection Models

Multi-stage object detection models involve a more complex process that typically includes distinct phases for generating region proposals and then classifying these proposals into specific object categories. These models first isolate potential object locations before applying sophisticated classification and bounding box regression techniques. This separation allows for more precise detections and higher overall accuracy, making multi-stage models adept at handling complex image scenarios with multiple objects and varying scales. The trade-off, however, is usually slower processing speeds compared to single-stage detectors.

2.1.1 Region-based Convolution Neural Networks

The R-CNN family of models represents a fundamental shift in object detection, introducing deep learning to generate high-quality region proposals that are then classified by a convolutional neural network. This evolutionary path not only streamlined the detection pipeline but also improved the scalability and applicability of these models in real-world scenarios, where speed and accuracy are crucial. The successive refinements from R-CNN through Mask R-CNN highlight a trajectory of continuous improvement, with each iteration bringing more sophisticated integration of features and functionalities.

Region-based Convolution Neural Networks

The need for more sophisticated solutions that could accurately identify and locate multiple objects within images led to the development of Region-based Convolutional Networks (R-CNNs). Starting with the base Region-based Convolution Neural Network This approach combines region proposal algorithms with the feature extraction capabilities of CNNs. R-CNNs begin by generating potential object-bound regions in an image, a process known as region proposal. Each region is then cropped and resized to a fixed size before being fed into a pre-trained convolutional neural network.

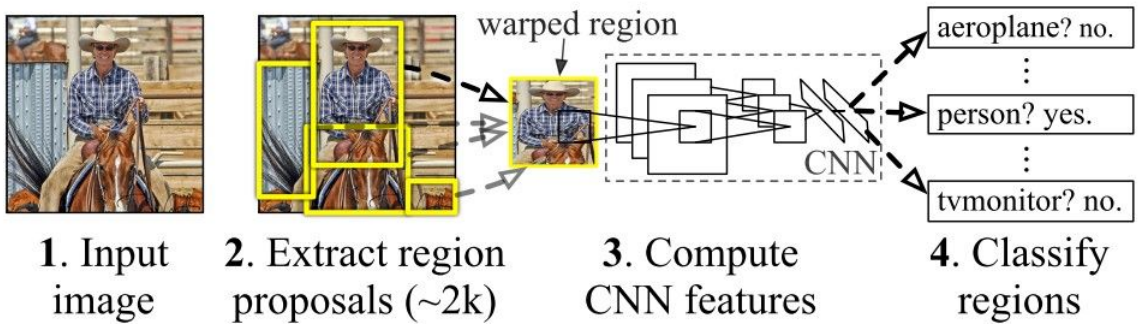

Figure 2.1: Region-based Convolution Neural Network Architecture

As presented in the 2.1 the R-CNN consists of 3 main modules. The first module generates 2,000 region proposals using the Selective Search algorithm. After being resized to a fixed pre-defined size, the second module extracts a feature vector of length 4,096 from each region proposal. The third module uses a pre-trained SVM algorithm to classify the region proposal to either the background or one of the object classes.

Some the limitations of the R-CNN model are the facts that it is a multi-stage model, where each stage is an independent component, thus, it cannot be trained end-to-end. Also the R-CNN depends on the Selective Search algorithm for generating region proposals, which takes a lot of time and cannot be customized to the detection problem. Lastly each region proposal is fed independently to the CNN for feature extraction, which makes it impossible to run R-CNN in real-time.

Fast Region-based Convolution Neural Networks

Fast R-CNN improved upon the original R-CNN's efficiency, where instead of cropping and resizing each region separately, the entire image is passed through the CNN to extract features. Regions of interest (ROIs) are then selected from the feature map using the proposed bounding boxes from the selective search. These ROIs are then pooled into a fixed-size feature map and passed through fully connected layers for classification and bounding box regression in the figure 2.2.

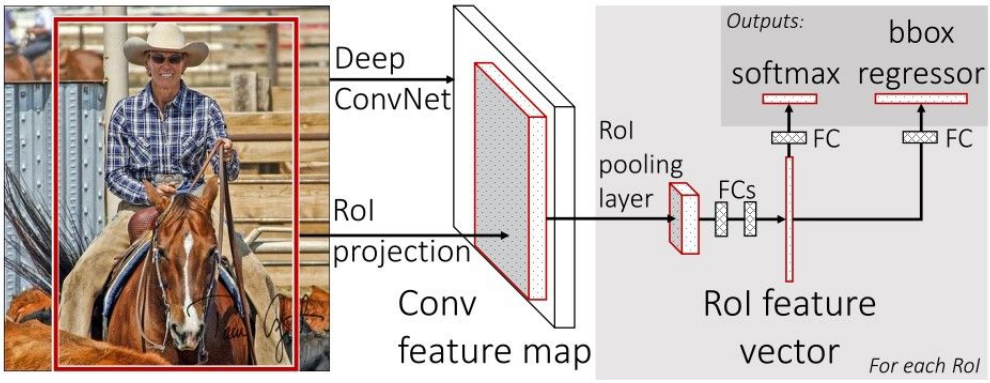

Figure 2.2: Fast Region-based Convolution Neural Network Architecture

In this model a proposed a new layer called ROI Pooling that extracts equal-length feature vectors from all proposals in the same image, where compared to R-CNN, which has multiple stages, Faster R-CNN builds a network that has only a single stage.

The RoI pooling layer uses max pooling to convert the features inside any valid region of interest into a small feature map with a fixed spatial extent of $H \times W$, where H and W are layer hyper-parameters that are independent of any particular RoI. In this paper, an RoI is a rectangular window into a convolution feature map. Each RoI is defined by a four-tuple (r, c, h, w) that specifies its top-left corner (r, c) and its height and width (h, w). Also one of the great inclusions of this model was the implementation of multi-task loss:

$$L(p, u, t', v) = L_{cls}(p, u) + \lambda[u \geq 1]L_{loc}(t', v)z \tag{1}$$

,where the classification loss $L_{cls}(p, u)$ is defined as the negative log likelihood of the true class u, expressed as:

$$L_{cls}(p, u) = -\log p_u \tag{2}$$

The localization loss L_{loc} is defined over the predicted bounding box parameters $t' = (t'_x, t'_y, t'_w, t'_h)$ and the ground truth bounding box parameters $v = (v_x, v_y, v_w, v_h)$ for class u. The Iverson bracket $[u \geq 1]$ is used as an indicator function that evaluates to 1 when u is 1 or more, and 0 otherwise. This function helps in applying the localization loss only when there is a foreground class detected, effectively ignoring the background.

The overall loss $L(p, u, t', v)$ is then a combination of classification and localization losses, modulated by a parameter λ, representing the trade-off between these two tasks:

$$L(p, u, t', v) = L_{cls}(p, u) + \lambda[u \geq 1]L_{loc}(t', v) \tag{3}$$

Faster Region-based Convolution Neural Networks

While Fast R-CNN improved upon its predecessors in terms of both speed and accuracy, the Faster R-CNN[4] architecture emerged as an even more refined version. Fast R-CNN effectively addressed the inefficiencies of previous models by integrating a region of interest (RoI) pooling layer to connect convolutional feature extraction and region proposal tasks. However, it still relied on external region proposal algorithms, which remained a bottleneck in terms of computational efficiency and speed.

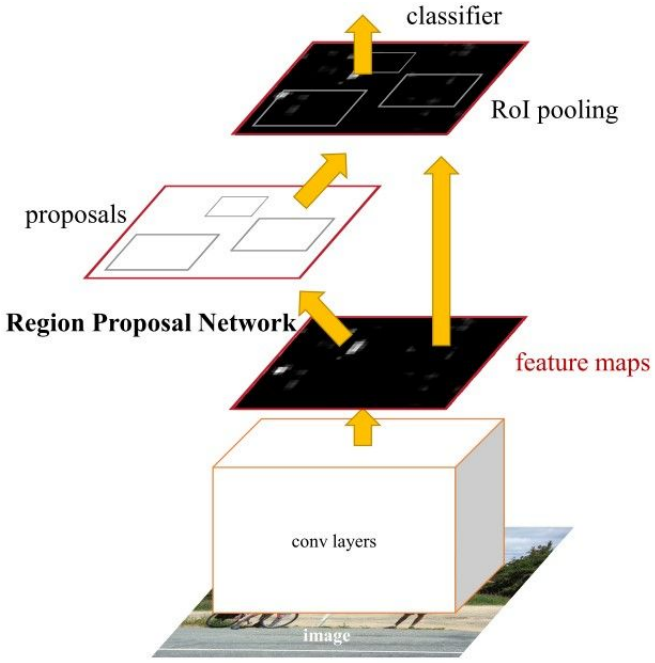

Figure 2.3: Faster Region-based Convolution Neural Network Architecture

Faster R-CNN resolved this by introducing a novel component, the Region Proposal Network (RPN)[5]. A Region Proposal Network takes an image (of any size) as input and outputs a set of rectangular object proposals, each with an objectness score. This process is modeled with a fully convolutional network, because the ultimate goal is to share its computation with a Fast R-CNN object detection network, as it is assumed that both nets share a common set of convolutional layers, as seen in the Figure 2.3.

To generate region proposals, a small network slides over the convolutional feature map output by the last shared convolutional layer. This small network takes as input an $n \times n$ spatial window of the input convolutional feature map. Each sliding window is mapped to a lower-dimensional feature. This feature is fed into two sibling fully-connected layers—a box-regression layer (reg layer) and a box-classification layer (cls layer). This mini-network is illustrated at a single position in the Figure 2.4. Also since the mini-network operates in a sliding-window fashion, the fully-connected layers are shared across all spatial locations.

This architecture is naturally implemented with an n×n convolutional layer followed by two sibling 1×1 convolutional layers.

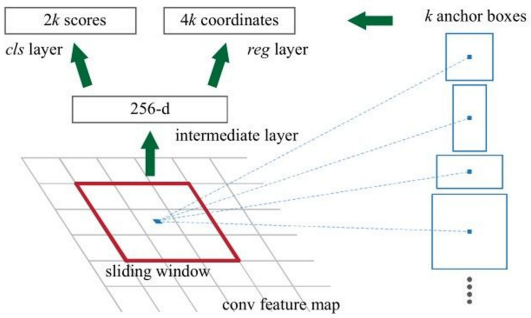

Figure 2.4: Region Proposal Network

At each sliding-window location, there is simultaneously a prediction of multiple region proposals, where the number of maximum possible proposals for each location is denoted as k. The k proposals are parameterized relative to k reference boxes, which are called anchors. An anchor is centered at the sliding window in question, and is associated with a scale and aspect ratio as seen again in Figure 2.4. By default 3 scales and 3 aspect ratios are used, yielding k = 9 anchors at each sliding position. For a convolutional feature map of a size $W \times H$, there are $W \times H \times k$ anchors in total.

Masked Region-based Convolution Neural Networks

Mask R-CNN builds on the ideas and successes of the Faster R-CNN model, which predicts both a class label and a bounding-box offset for each candidate object. To these, Mask R-CNN adds a third branch specifically designed to output the object mask, providing a straightforward and logical extension to the existing framework. This addition allows Mask R-CNN to capture the precise spatial layout of objects, a task that necessitates extracting significantly finer detail than what is required for classifying objects or predicting bounding boxes alone.

Mask R-CNN adopts the same two-stage procedure as Faster R-CNN, with an identical first stage the RPN. In the second stage, in parallel to predicting the class and box offset, Mask R-CNN also outputs a binary mask for each RoI. This is in contrast to most recent systems, where classification depends on mask predictions. This approach follows the spirit of Fast R-CNN that applies bounding-box classification and regression in parallel.

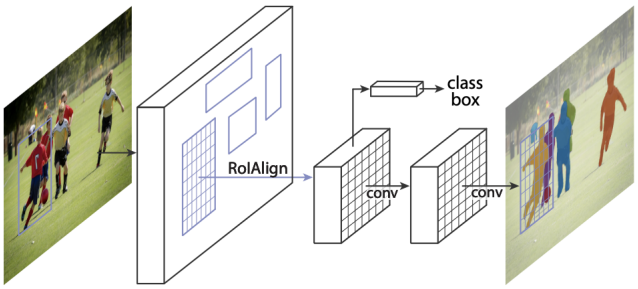

Figure 2.5: Mask Region Convolution Neural Network Architecture

A mask encodes an input object's spatial layout. Thus, unlike class labels or box offsets that are inevitably collapsed into short output vectors by fully-connected layers, extracting the spatial structure of masks can be addressed naturally by the pixel-to-pixel correspondence provided by convolutions. Specifically, an $m \times m$ mask is predicted from each RoI using an FCN. This allows each layer in the mask branch to maintain the explicit $m \times m$ object spatial layout without collapsing it into a vector representation that lacks spatial dimensions. Unlike previous methods that resort to fully-connected layers for mask prediction, this fully convolutional representation requires fewer parameters, and is more accurate as demonstrated by experiments.

This pixel-to-pixel behavior requires our RoI features, which themselves are small feature maps, to be well aligned to faithfully preserve the explicit per-pixel spatial correspondence. This motivated us to develop the following RoIAlign layer that plays a key role in mask prediction.

RoIPool is a standard operation for extracting a small feature map like 7×7 from each RoI. RoIPool first quantizes a floating-number RoI to the discrete granularity of the feature map, this quantized RoI is then subdivided into spatial bins which are themselves quantized, and finally feature values covered by each bin are aggregated usually by max pooling.

Quantization, such as that performed on a continuous coordinate x by calculating $\frac{x}{16}$, where 16 represents the feature map stride accompanied by the rounding. Similarly, this quantization process occurs when coordinates are divided into discrete bins, for example, into a $7x7$ grid. However, these quantization steps can lead to slight misalignments between the Region of Interest (RoI) and the features extracted from it, potentially affecting the accuracy of the model.

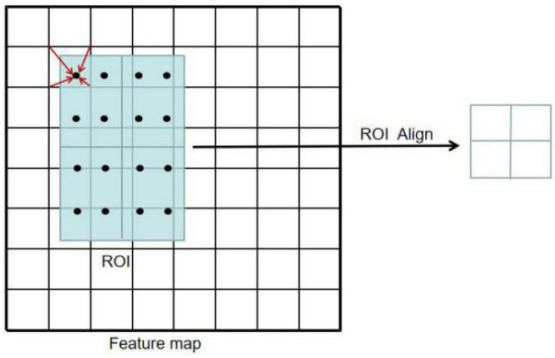

Figure 2.6: Mask Region Of Interest

While this may not impact classification, which is robust to small translations, it has a large negative effect on predicting pixel-accurate masks. To address this, a RoIAlign layer is proposed that removes the harsh quantization of RoIPool, properly aligning the extracted features with the input. A bilinear interpolation is used to compute the exact values of the input features at four regularly sampled locations in each RoI bin, and aggregate the result.

Furthermore once again this model utilizes a multi-task loss in order to take into consideration all the outputs of the model.

$$L = L_{cls} + L_{box} + L_{mask} \tag{4}$$

The mask branch has a $K \times m^2$-dimensional output for each RoI, which encodes K binary masks of resolution m $\times$ m, one for each of the K classes. To this a per-

pixel sigmoid is applied, and define Lmask as the average binary cross-entropy loss. For an RoI associated with ground-truth class k, L_{mask} is only defined on the k-th mask. This definition of L_{mask} allows the network to generate masks for every class without competition among classes, since on the dedicated classification branch to predict the class label used to select the output mask. This decouples mask and class prediction. This is different from common practice when applying FCNs to semantic segmentation, which typically uses a per-pixel softmax and a multinomial cross-entropy loss. In that case, masks across classes compete; in our case, with a per-pixel sigmoid and a binary loss, they do not.

2.1.2 Feature Pyramid Networks

The concept of Feature Pyramid Networks (FPNs) was a significant advancement in object detection and segmentation, by leveraging the inherent multi-scale, pyramidal hierarchy of deep convolutional networks. The first subsection details the basic Feature Pyramid Network, an architecture that efficiently creates a pyramid of features at multiple levels, enabling the detection of objects at various scales with improved accuracy. This structure uses a top-down approach with lateral connections to combine low-resolution, semantically strong features with high-resolution, semantically weaker features, enhancing the network's ability to detect objects across different scales. The subsequent subsection explores the Extended Feature Pyramid Network, which builds upon the original FPN by introducing enhancements that further refine feature integration and improve detection performance, particularly in challenging scenarios with complex object orientations and scales.

Feature Pyramid Network

With Feature Pyramid Networks[] the objective is to harness the pyramidal feature hierarchy of a Convolutional Networks, which inherently contains semantic information from low to high levels, to construct a feature pyramid that maintains high-level semantics throughout. The method processes a single-scale image of arbitrary size and produces proportionally sized feature maps at multiple levels, utilizing a fully convolutional approach. This procedure is compatible with various backbone convolutional architectures, and this paper demonstrates results using ResNets. The construction of the pyramid incorporates a bottom-up pathway, a top-down pathway, and lateral connections, which are detailed below and seen in the Figure 2.7.

The bottom-up pathway is the feed forward computation of the ConvNet backbone, which computes a feature hierarchy consisting of feature maps at several scales with a scaling step of 2. There are often many layers producing output maps of the same size and these layers are in the same network stage. For the feature pyramid, one pyramid level for each stage is defined. The output of the last layer of each stage is chosen as the reference set of feature maps, which will be enriched to create the pyramid. This choice is logical because the deepest layer of each stage is expected to have the strongest features.

Specifically, for ResNets, the feature activations output by the last residual block of each stage are utilized. These outputs are denoted as $C2, C3, C4, C5$ corresponding to the outputs of conv2, conv3, conv4, and conv5, respectively. It is important to note that these have strides of $4, 8, 16, 32$ pixels with respect to the input image. The conv1 output is not included in the pyramid due to its large memory footprint.

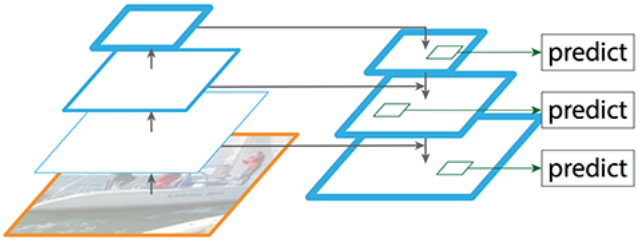

Figure 2.7: Feature Pyramid Network

The top-down pathway hallucinates higher resolution features by up-sampling spatially coarser, but semantically stronger, feature maps from higher pyramid levels. These features are then enhanced with features from the bottom-up pathway via lateral connections. Each lateral connection merges feature maps of the same spatial size from the bottom-up pathway and the top-down pathway. The bottom-up feature map is of lower-level semantics, but its activations are more accurately localized as it was sub-sampled fewer times. With a coarser-resolution feature map, the spatial resolution is up-sampled by a factor of 2. The up-sampled map is then merged with the corresponding bottom-up map (which undergoes a 1×1 convolutional layer to reduce channel dimensions) by element-wise addition as seen in the figure 2.8. This process is iterated until the finest resolution map is generated. To start the iteration, simply attach a 1×1 convolutional layer on $C5$ to produce the coarsest resolution map.

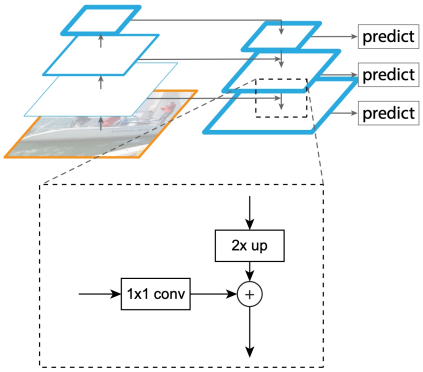

Figure 2.8: Feature Pyramid Network Top-Down path

Finally a 3×3 convolution is added on each merged map to generate the final feature map, which is to reduce the aliasing effect of up-sampling. This final set of feature maps is called $P2, P3, P4, P5$, corresponding to $C2, C3, C4, C5$ that are respectively of the same spatial sizes. Since all levels of the pyramid use shared classifiers/regressors as in a traditional featurized image pyramid, a fixed feature dimension or numbers of channels is set, with the value being denoted as d in all the feature maps. With $d = 256$ in this paper and thus all extra convolutional layers have 256-channel outputs. Simplicity is central to this design and it has been proven that this model is robust to many design choices. Experiments with more sophisticated blocks have been made and observed marginally better results.

2.1.3 Vision Transformers

The Vision Transformer (ViT)[] model represents a significant shift in the approach to image recognition, applying architectures originally developed for natural language processing (NLP) to computer vision tasks. The ViT moves away from traditional convolutional neural networks (CNNs) to embrace transformers, a model type that leverages self-attention mechanisms originally designed for text data. Like NLP systems, ViT treats an image as a sequence of fixed-size patches, akin to words in a sentence, and processes these patches through a series of transformer layers to capture complex inter-patch relationships and contextual information.

In a transformer model for NLP, the encoder processes the input text by mapping it into a high-dimensional space to capture contextual relationships, while the decoder generates the output text sequentially, using the encoder's context-rich representations along with its own input to predict the next word or sequence, thereby facilitating tasks like translation, summarization, and text generation. The architecture of a transformer model can be seen in the figure 2.9, where the left part is the transformer encoder and the right part is the decoder.

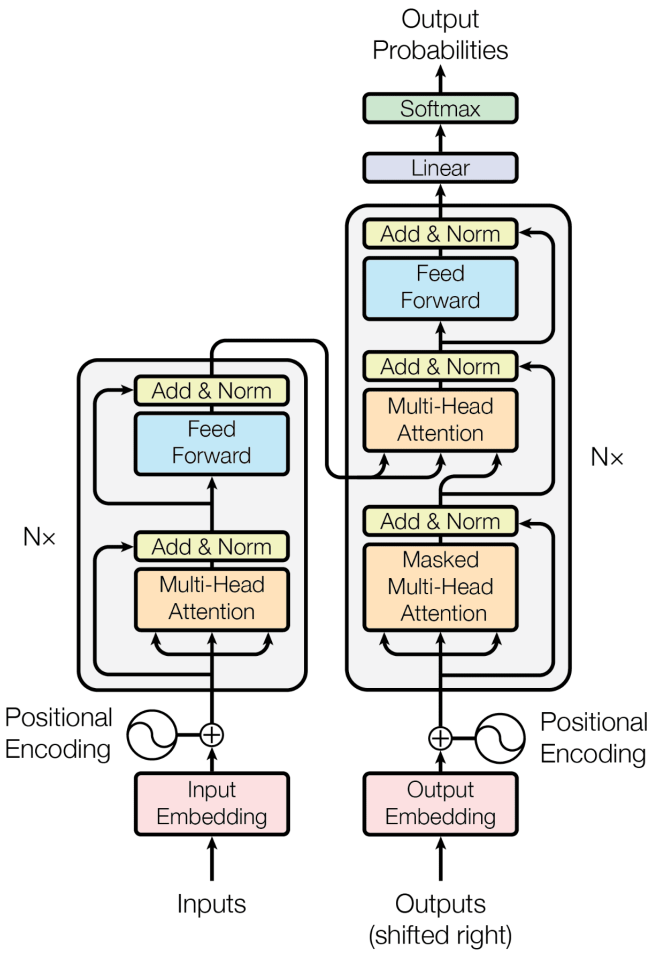

Figure 2.9: NLP Transformer Architecture

On the other hand, the ViT encoder processes the input image, which is divided into patches and linearly embedded. The encoder's self-attention mechanism allows the model to consider each patch in the context of others, capturing the global context of the image. This is crucial for tasks like image classification or object detection, where understanding the entire image context is important. The encoder is responsible for understanding the content and context of the image, learning to identify features and patterns relevant to the task at hand. Furthermore ViTs typically do not use a decoder because their primary tasks do not involve generating sequential data as is the case in machine translation or text generation in NLP. Instead, the output of the ViT encoder is directly connected to task-specific heads like linear layers for classification or additional layers for object detection that interpret the encoder's representations to make predictions.

In the pre-transformer encoder stage of processing an image, the initial input, typically of size $H \times W \times D$ (height, width, depth), is segmented into N non-overlapping patches. Each patch has dimensions $P \times P \times D$ and is extracted in a grid-like pattern across the image. These patches are then flattened and undergo a linear projection to transform them into a higher-dimensional space, with each patch represented as a one-dimensional array. To ensure that spatial information is not lost during this transformation, positional encodings of the same dimension, D', are added to these patch embeddings. This results in each patch embedding maintaining a sequence of vectors, each with dimension D'. The linear projection itself is facilitated by a learnable matrix that the flattened patches are multiplied with, culminating in an array of size $N \times D'$, where each row represents the embedded representation of a patch. This structured approach allows the model to retain a detailed understanding of the original image's spatial layout as it processes through the transformer encoder.

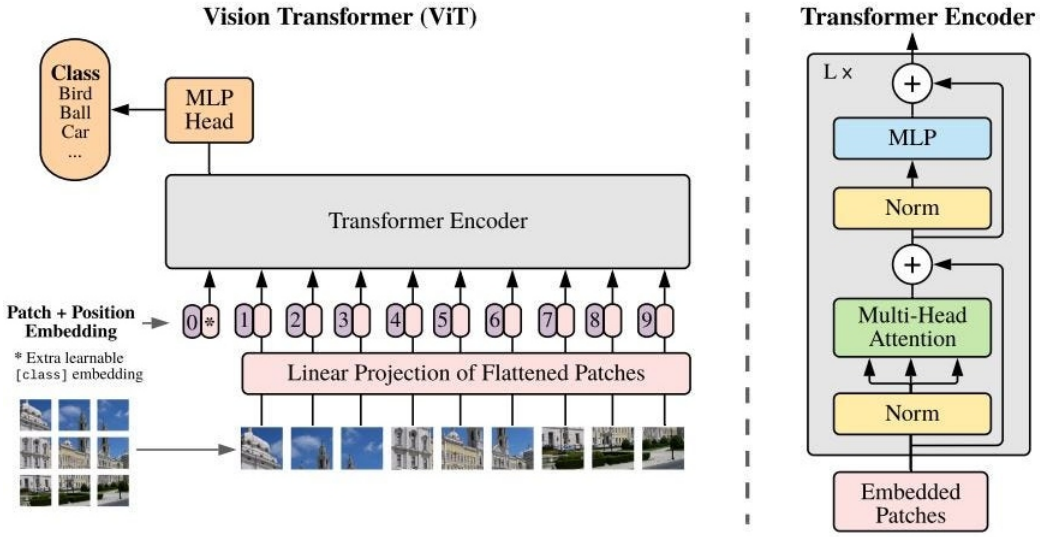

Figure 2.10: Vision Transformer Architecture

The transformer encoder in image processing is composed of several transformer blocks arranged in sequence, each designed to refine the representation of image patches through a series of specialized sub-components. Each block starts with Layer Normalization, which is applied before self-attention and MLP operations to stabilize the training process and facilitate model convergence. Unlike batch normalization, layer normalization is preferred in transformers as they handle sequential data, ensuring that the size of the data remains at $N \times D'$.

Following normalization, the Multi-Head Self-Attention (MHA) mechanism is used, accepting queries, keys, and values as inputs. This allows the model to simultaneously focus on different parts of the image, enhancing the contextual understanding of the image by computing attention scores and updating embeddings without altering the initial data size of $N \times D'$.

Residual Connections are employed after MHA and MLP within each transformer block, where the input to the block is added back to its output. This practice helps preserve information across the layers, combating the vanishing gradient problem and facilitating deeper model architectures by improving gradient flow.

Additionally, each block contains an MLP (Multi-Layer Perceptron) segment, which includes fully connected layers featuring a GELU activation function and often incorporates dropout to prevent overfitting. This component further processes the output of the self-attention mechanism before it passes through another round of residual connections.The cumulative effect of these operations in each transformer block is to continuously refine the embeddings, with the input and output of each block being arrays of size $N \times D'$, effectively updating the representation at each step of the sequence.

In the post-transformer encoder phase of image processing, several critical modules refine the processed features for specific tasks such as object detection and classification. The MLP Head is employed immediately after the transformer encoder to project the encoded features into a lower-dimensional space that is more suitable for object detection tasks. The dimensionality of the MLP head's output varies depending on its architecture and the specifics of the object detection task being performed, adapting the complex, high-dimensional data into a form that can be effectively used for the subsequent steps.

2.2 Single-stage Object Detection Models

Single-stage object detection models represent a streamlined approach to identifying and localizing objects within an image. Unlike multi-stage models, single-stage models, such as YOLO (You Only Look Once) and SSD (Single Shot MultiBox Detector), operate in a direct manner by predicting object classes and bounding box coordinates in a single forward pass through the network. This methodology not only simplifies the detection pipeline but also enhances the speed of detection, making single-stage models particularly well-suited for real-time applications. However, this increase in speed can sometimes come at the cost of lower accuracy, especially in detecting small or overlapping objects.

2.2.1 You Only Look Once

YOLO (You Only Look Once)[] represents a shift in object detection techniques through its simple and highly efficient approach. Unlike traditional object detection methods that involve a multi-step process, YOLO employs a single convolutional network that simultaneously predicts multiple bounding boxes and class probabilities for those boxes. This process is followed by a non-maximum suppression step to finalize the detections. The architecture is based on a standard high-quality image classification network truncated before any classification layers, known as the base network, which is then augmented with additional layers to detect objects at multiple scales.

One of the primary benefits of YOLO is its speed, achieving real-time processing rates of up to 45 frames per second on a Titan X GPU, and a fast version that exceeds 150 fps. This capability allows YOLO to handle streaming video in real-time with close to no latency, significantly outperforming other real-time systems in mean average precision. Moreover, YOLO reasons globally about the image during both training and testing phases, which helps it encode contextual information about object classes and their appearances. This global perspective reduces background errors significantly compared to methods like Fast R-CNN, which often misclassify background patches as objects.

In addition, YOLO learns generalizable representations of objects, demonstrated by its superior performance when trained on natural images and tested on artwork, outperforming other top detection methods. However, while YOLO excels in identification speed and generalizability, it sometimes struggles with precisely localizing objects, particularly small ones.

The network divides the input image into an $S \times S$ grid, with each grid cell responsible for detecting objects whose centers fall within the cell. Each cell predicts multiple bounding boxes and confidence scores, which reflect the presence and accuracy of object detection. These predictions are based on the intersection over union (IOU) metric between the predicted boxes and ground truth.

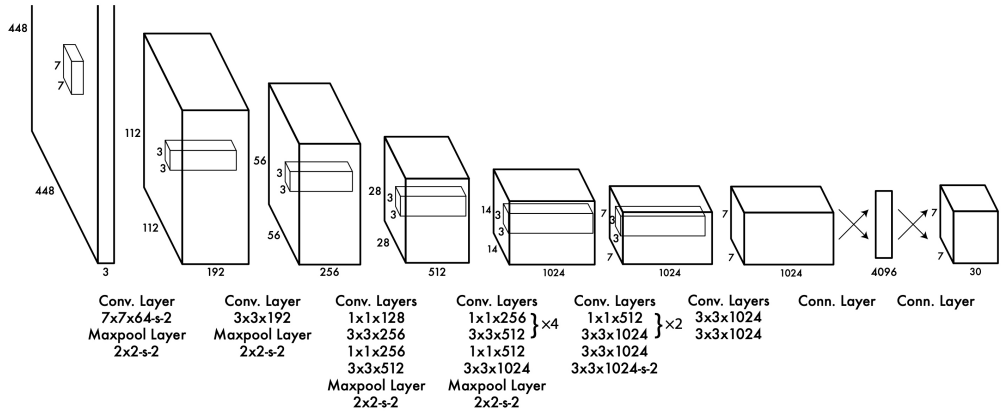

Figure 2.11: YOLO Architecture

The convolutional neural network of YOLO, inspired by the GoogLeNet model for image classification, comprises 24 convolutional layers followed by 2 fully connected layers. This structure utilizes 1×1 reduction layers followed by 3×3 convolutional layers to efficiently predict bounding boxes and class probabilities. Despite its strengths, YOLO imposes certain spatial constraints on bounding box predictions due to each grid cell predicting only two boxes and one class, which can limit its effectiveness in scenarios with small or clustered objects. Additionally, YOLO's simpler architecture struggles with unusual object aspect ratios and configurations and is less forgiving of localization errors in smaller bounding boxes due to its coarse feature predictions and the uniform treatment of errors across all box sizes in its loss function.

In conclusion, YOLO's innovative approach integrates the components of object detection into a single neural network, leveraging features from the entire image to make global predictions about all objects present. This end-to-end training capability and real-time processing speed maintain high average precision, making YOLO a groundbreaking model in the field of object detection, despite some challenges with small and closely spaced objects.

2.2.2 Single Shot MultiBox Detector

The Single Shot MultiBox Detector (SSD)[] employs a streamlined architecture that integrates detection and classification into a single pass through a feed-forward convolutional network, thereby enhancing speed without compromising accuracy. This model produces a fixed collection of bounding boxes and class scores for each object instance identified, followed by a non-maximum suppression step to finalize the detections. The foundation of SSD is a base network derived from standard high-quality image classification architectures, which is truncated before any classification layers.

To adapt this base network for object detection, SSD incorporates multiple convolutional feature layers at the end of the network. These additional layers decrease progressively in size, allowing for the detection of objects at various scales—a notable improvement over models like Overfeat and YOLO, which are limited to single scale feature maps. Each feature layer in SSD employs a unique set of convolutional filters to predict detection outcomes at that scale, thus enhancing the model's responsiveness to objects of varying dimensions as seen in the Figure 2.12.

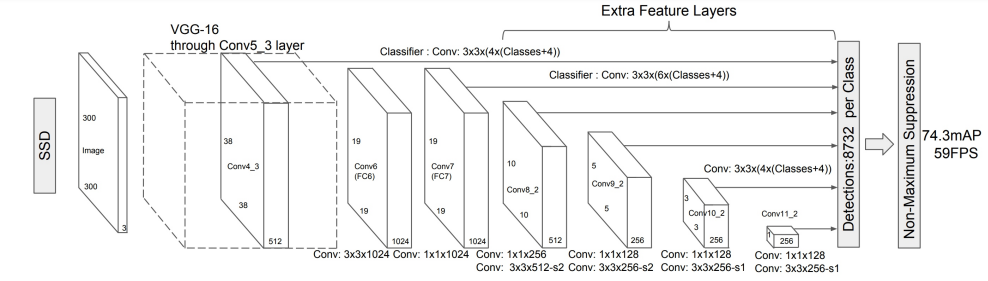

Figure 2.12: Single Shot MultiBox Detector Architecture

SSD further refines its detection capabilities through the use of convolutional predictors at each feature layer. These predictors apply a 3×3×p convolutional kernel to each feature map location, generating scores for category presence or box shape offsets relative to predefined default boxes. These default boxes, similar to the anchor boxes used in Faster R-CNN but applied across multiple resolutions, are associated with each feature map cell. For each default box, the model computes class scores and spatial offsets, improving detection precision. This mechanism results in an extensive set of outputs for each feature map cell, ensuring robust detection across a diverse array of object types and sizes.

3 Methodology

In our exploration of various object detection architectures, it became clear that single-stage detector models, while notable for their efficiency, did not meet the accuracy demands required for small object detection. Given the challenges associated with detecting smaller objects, we considered multi-stage models as a more viable solution, because these models are known for their enhanced precision and adeptness at handling intricate detection tasks. Through our research, a potential model combination emerged as a great candidate. The Feature Pyramid Network family of models excels in multi-scale representation essential for resolving objects of varying sizes, while the Vision Transformers family provides an exceptional ability to grasp global contextual dependencies within images through the attention modules.

By integrating these models, as a backbone and a detector we found that we can leverage the complementary strengths of both frameworks, thus creating a robust multi-stage model specifically made to achieve the best possible results in small object detection.

3.1 Architecture

Choosing a multi-stage model for small object detection does not necessarily imply that the model will be heavy in terms of trainable parameters or challenging to train. Our approach strategically combines the two types of models in order to harness their individual strengths while maintaining manageable complexity and training efficiency. This integration ensures that our multi-stage detector not only achieves an elevated accuracy but also remains practical in terms of computational resources and training time, making it a viable solution for real-world applications where both performance and efficiency are critical.

In this section we are going to describe and explain the model we developed in detail with the aim of successfully tackling the Small Object Detection challenge. The model we created is named Extended Masked-Attention Mask Transformer, a name based on the models we decided to integrate being the Extended Feature Pyramid Network[] and the Masked-Attention Mask Transformer (MAMT)[].

A very interesting finding in the EFPN article is that over the past years, rapid development of deep learning has boosted the popularity of CNN-based detectors and the improved overall accuracy and efficiency. Unfortunately they still perform poorly when detecting small objects with a few pixels. Since CNN uses pooling layers repeatedly to extract advanced semantics, the pixels of small objects can be filtered out during the down-sampling process.

Utilization of low-level features is one way to pick up information about small objects. The FPN is the first method to enhance features by fusing features from different levels and constructing feature pyramids, where upper feature maps are responsible for larger object detection, and lower feature maps are responsible for smaller object detection. Despite FPN's improvement multi-scale detection performance, the heuristic mapping mechanism between pyramid level and proposal size in FPN detectors may confuse small object detection, since small-sized objects must share the same feature map with medium-sized objects and some large-sized objects.

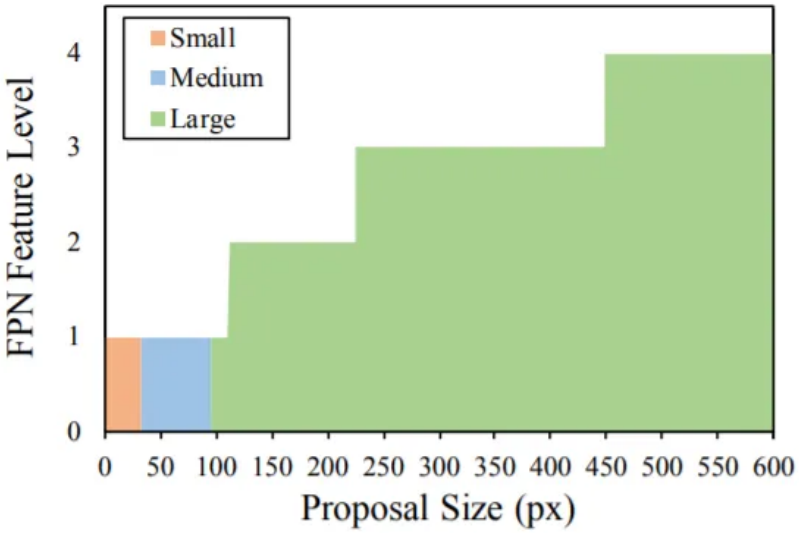

Figure 3.1: Small Object Detection Mapping

3.1.1 Extended Feature Pyramid Network

The implementation of the Extended Feature Pyramid follows closely an FPN-like framework embedded with a feature Super Resolution (SR)[] module. This pipeline directly generates high resolution features from low-resolution images to support small object detection, while stays in low computational cost. The top four pyramid layers are constructed by top-down pathways for medium and large object detection.

The bottom extension of the EFPN contains the very important FTT module, the top-down pathway and the purple pyramid layer in 3.2, that aims to capture regional details for small objects. More specifically, in the extension, the third and fourth pyramid layers of EFPN which are denoted by green and yellow layers respectively in 3.2, are mixed up in the feature super resolution module, the FTT with the aim of producing the intermediate feature P_3' with selected regional information, which is denoted by a blue diamond in Figure 3.2. Finally the top-down pathway merges P_3' with a tailor-made high-resolution CNN feature map $C'2$, producing the final extended pyramid layer $P'2$.

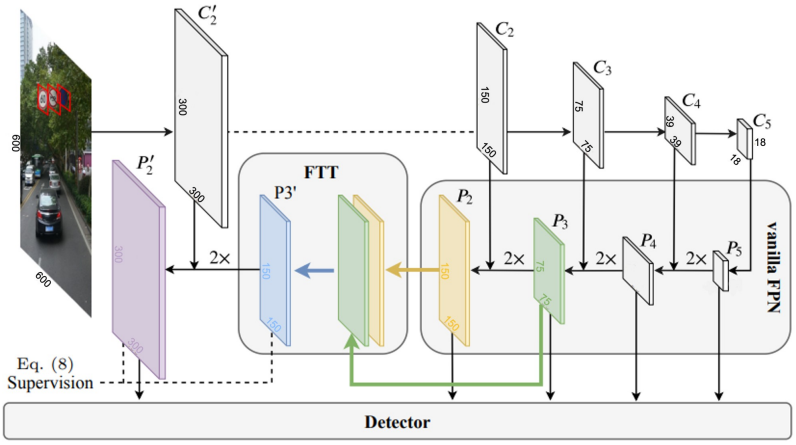

Figure 3.2: Extended Feature Pyramid Network

The current Feature Texture Transfer (FTT) output as seen in the figure synthesizes strong semantics in upper low-resolution features and critical local details in lower high-resolution reference features, but discards disturbing noises in reference. As shown in Figure 3.3, the main input of FTT module is the feature map P3 from the 3rd layer of EFPN, and the reference is the feature map P2 from the 4th layer of EFPN.

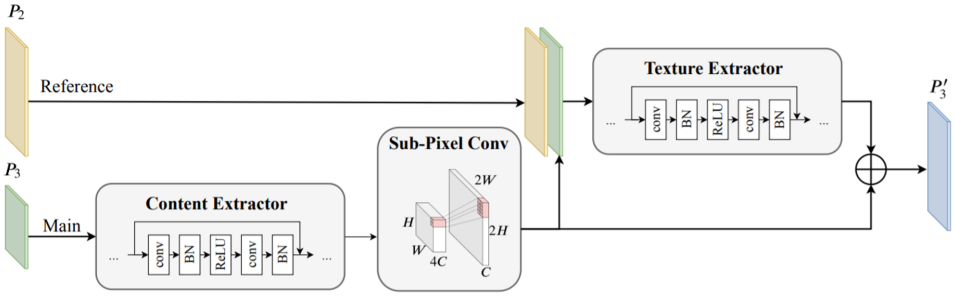

Figure 3.3: Feature Texture Transfer

The output P_3' can be defined as:

$$P_3' = E_t(P_2 \| E_c(P_3)_{\uparrow 2}) + E_c(P_3)_{\uparrow 2} \tag{5}$$

,where where $E_t()$ denotes texture extractor component, $E_c()$ denotes content extractor component, $\uparrow 2\times$ here denotes double up-scaling by sub-pixel convolution and k denotes feature concatenation. The content extractor and texture extractor are both composed of residual blocks. In the main stream, a sub-pixel convolution is applied to upscale spatial resolution of the content features from the main input $P3$ considering its efficiency. Sub-pixel convolution augments pixels on the dimensions of width and height via diverting pixels on the dimension of channel.

3.1.2 Masked-Attention Mask Transformer

Mask classification architectures group pixels into N segments by predicting N binary masks, along with N corresponding category labels. Mask classification is general enough to address any segmentation or object detection task by assigning different semantics. However, the challenge is to find good representations for each segment. Inspired by DETR [], each segment in an image can be represented as a C-dimensional feature vector know as object query and can be processed by a Transformer decoder, trained with a set prediction objective. A simple meta architecture would consist of three components.

A backbone that extracts low resolution features from an image. A pixel decoder that gradually up-samples low-resolution features from the output of the backbone to generate high-resolution per-pixel embeddings. And finally a Transformer decoder that operates on image features to process object queries. The final binary mask predictions are decoded from per-pixel embeddings with object queries. One successful instantiation of such a meta architecture is MaskFormer [].

Mask2Former adopts this kind meta architecture, with the proposed Transformer decoder in the Figure 3.4 replacing the standard one. The key components of the Transformer decoder include a masked attention operator, which extracts localized features by constraining cross attention to within the foreground region of the predicted mask for each query, instead of attending to the full feature map.

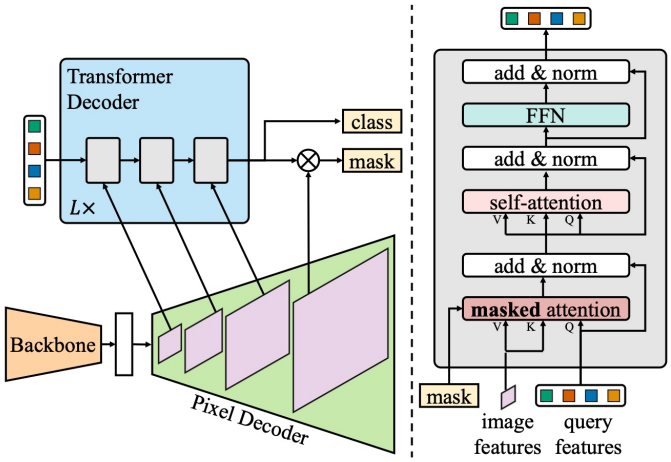

Figure 3.4: Masked-Attention Mask Transformer

To handle small objects, an efficient multi-scale strategy is proposed to utilize high-resolution features. It feeds successive feature maps from the pixel decoder's feature pyramid into successive Transformer decoder layers in a round robin fashion. Context features have been shown to be important for image segmentation. However, recent studies [] suggest that the slow convergence of Transformer-based models is due to global context in the cross-attention layer, as it takes many training epochs for cross-attention to learn to attend to localized object regions.

There is the hypothesize that local features are enough to update query features and context information can be gathered through self-attention. For this a masked attention was proposed, a variant of cross attention that only attends within the foreground region of the predicted mask for each query. Standard cross-attention computes:

$$X_l = \text{softmax}(Q_l K_l^T) V_l + X_{l-1} \tag{6}$$

Here, l is the layer index, $X_l \epsilon R^{N \times C}$ refers to N C-dimensional query features at the lth layer and $Q_l = f_Q(X_l - 1) \epsilon R^{N \times C}$. Here X_0 denotes input query features to the Transformer decoder. $K_l, V_l \epsilon R^{H_l W_l \times C}$ are the image features under transformation $f_K()$ and $f_V()$ respectively, and H_l and W_l are the spatial resolution of image features, f_Q, f_K and f_V are linear transformations

Our masked attention modulates the attention matrix via:

$$X_l = \text{softmax}(\mathcal{M}_{l-1} + Q_l K_l^T) V_l + X_{l-1} \tag{7}$$

Moreover, the attention mask M_{l-1} at feature location (x, y) is:

$$\mathcal{M}_{l-1}(x, y) = \begin{cases} 0 & \text{if } \mathcal{M}_{l-1}(x, y) = 1 \\ -\infty & \text{otherwise} \end{cases} \tag{8}$$

Here, $M_{l-1} \epsilon \{0, 1\}^{N \times H_l W_l}$ is the binarized output, thresholded at 0.5 of the resized mask prediction of the previous $(l-1)$-th Transformer decoder layer. It is resized to the same resolution of K_l. M_0 is the binary mask prediction obtained from X_0, before feeding query features into the Transformer decoder.

High-resolution features improve model performance, especially for small objects [], however, this is computationally demanding. Thus the solution being an efficient multi-scale strategy to introduce high-resolution features while controlling the increase in computation. Instead of always using the high-resolution feature map, a feature pyramid which consists of both low- and high-resolution features is utilized and feed one resolution of the multi-scale feature to one Transformer decoder layer at a time. Specifically, the feature pyramid produced by the pixel decoder with resolution $1/32, 1/16$ and $1/8$ of the original image. For each resolution, both a sinusoidal positional embedding $e_{pos} \epsilon R^{HlWlC}$ and a learnable scale-level embedding $e_{lvl} \epsilon R^{1 \times C}$ is added.

We use those, from lowest-resolution to highest-resolution for the corresponding Transformer decoder layer as shown in Figure 3.4. This is repeated for a 3-layer Transformer decoder L times. In this case the final Transformer decoder hence has 3L layers.

More specifically, the first three layers receive a feature map of resolution $H1 = H/32, H2 = H/16, H3 = H/8, W1 = W/32, W2 = W/16, W3 = W/8$, where H and W are the original image resolution. This pattern is repeated in a round robin fashion for all following layers.

3.1.3 Extended Masked-Attention Mask Transformer

Throughout our extensive research into advanced object detection models and architectures, it became an interesting idea that a hybrid approach would yield the most effective results, especially for the task of small object detection. The detailed analysis of various architectures led us to the strategic combination of two highly effective models: the Extended Feature Pyramid Network (EFPN) and the Masked-Attention Mask Transformer (MAMT). This integration leverages the EFPN's robust capability in generating precise multi-scale feature maps and initial bounding boxes and masks with great efficiency, with the MAMT's advanced attention mechanisms that refine these masks for superior accuracy that is vital for the Small Object Detection task.

In this section we are going to analyze our model, the Extended Masked-Attention Mask Transformer or EMAMT starting with the models architecture in the Figure 3.5.

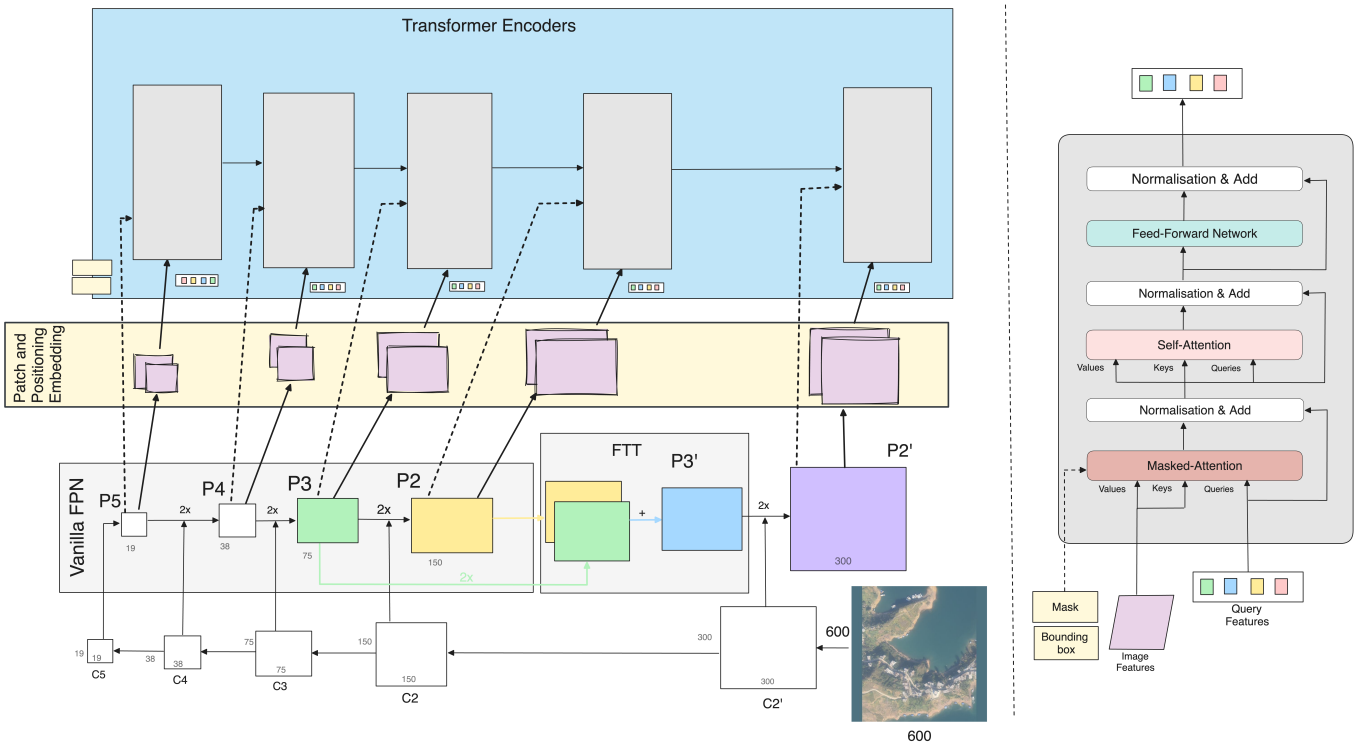

Figure 3.5: Extended Masked-Attention Mask Transformer

Firstly we are going to analyze the EFPN part of the model in the lower left part of the Figure 3.5, where the EFPN model has as a input, an image of size 600×600. The reason for the selection of that size is the backbone we use to generate the rich feature maps for the model. In our architecture we decided that a pre-trained feature extractor that combines great performance and light design was vital. This combination of characteristics led us to the choice of EfficientNet [] model and more specifically the EfficientNet-B7 model that uses an input size of 600×600. Furthermore we decided to create five feature maps that have the sizes $(300, 300)$, $(150, 150)$, $(75, 75)$, $(38, 38)$, $(19, 19)$ in order to have a wide range of data for the model to use.

Alongside the feature maps the EFPN also produces the bounding boxes and the initial binary masks that are going to be used by the MAMT detector. The bounding boxes and the masks are implemented on the most detailed feature map $P'2$ that is the result of the addition of the richest spatial map $C'2$ and the richest feature map created by the feature

texture transfer $P'3$. At the same time we have included a anchor generator model that creates anchors, informed based on the mean and standard deviation of the size of the bounding boxes of each dataset in order to conclude the first cycle of detection, between the actual data, the anchors and the bounding boxes. In this way we are able to train the EFPN model in parallel to the MAMT model, achieving better convergence. With the end of the EFPN model we are expecting a set of five feature maps, a set of bounding boxes and the corresponding classed for each object and the initial binary masks.

The next part of the EMAMT model is the patch and positional embedding we use in order to add to the model the sense of spatial arrangement among the input image patches. Since transformers inherently lack any mechanism to recognize order or position, positional embeddings are crucial for providing this spatial context. The creation of the positional embeddings follows the exact same idea as the original paper of MAMT, since it was the implementation of the sinusoidal positional embedding for images, adapted from the concept originally used in the transformer architecture for natural language processing. This type of embedding generates position encodings based on sine and cosine functions of different frequencies.

At this point we have a set of five embeddings of the feature maps, a set of bounding boxes and the corresponding classes and the binary masks. Along all that information the detector of this model we use a set number of queries that in this case is set to 100, because we wanted to focus on reliable detecting a sensible number of objects. For images that contain more than 100 objects and the annotations we decided to only use the first 100 of them to train the model. Provided all of this data the Vision Transformer architecture can be seen in the Figure 3.6.

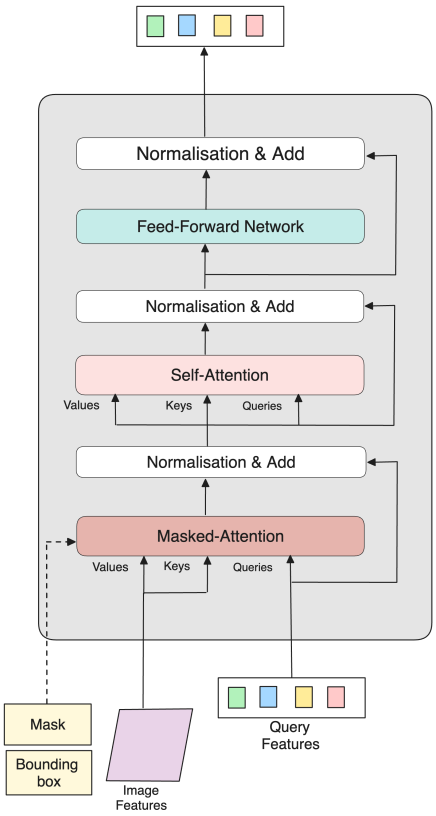

Figure 3.6: Extended Masked-Attention Mask Transformer - Transformer Module

As mentioned in the section 3.1.2 we have the hypothesis that local features are enough to update query features and context information can be gathered through self-attention. For this reason we use masked attention, a variant of cross-attention that only attends within the foreground region of the predicted mask for each query with the aim of better performance and faster convergence. More specifically the masked attention layer is used to focus the model's attention selectively based on certain predefined or learned criteria like the object queries, typically aligning with relevant areas of the input image. For instance, in tasks like image segmentation, the masked attention layer might concentrate processing on areas within a specific region of interest (ROI), such as the foreground of an image versus the background. This selective attention helps the model to efficiently process only the most informative parts of the image, thereby optimizing computational cost.

Following the masked attention, the cross-attention layer has to integrate additional context or features from another source into the transformer's workflow. When placed after a masked attention layer, the cross-attention layer can leverage the focused localized features processed by the masked attention and enrich these features with further contextual or supplementary data.

Lastly after the attention mechanisms—both masked and cross-attention—the FFN acts as a point of non-linear transformation that further processes the refined, attention-focused features. This structure allows the FFN to introduce additional complexity and abstraction capabilities into the feature representations.

In order to maximize the capabilities of this model we had to create a custom loss function that takes into consideration all the data created by the model. The loss function for the Extended Mask2Former model is designed to jointly optimize object detection and instance segmentation tasks by integrating multiple loss components. The total loss L_{total} is computed as a weighted sum of the mask loss, bounding box regression loss, and classification loss:

$$L_{\text{total}} = \lambda_{\text{mask}} L_{\text{mask}} + \lambda_{\text{bbox}} L_{\text{bbox}} + \lambda_{\text{class}} L_{\text{class}} \tag{9}$$

where:
λ_{mask}, λ_{bbox}, and λ_{class} are weighting coefficients for each loss component.

1. **Mask Loss**:
   The mask loss measures the discrepancy between the predicted masks $\hat{M}$ and the ground truth masks M. It uses the Binary Cross-Entropy Loss with logits:

$$L_{\text{mask}} = \frac{1}{N} \sum_{i=1}^{N} \text{BCEWithLogits}\left(\hat{M}_i,\, M_i\right) \tag{10}$$

   - N is the number of matched instances.
   - The matching between predicted and ground truth masks is performed using the Hungarian algorithm based on a combined cost of mask Intersection over Union (IoU) and classification cost.

2. **Matching via Hungarian Algorithm**:

   To align predicted instances with ground truth instances, the Hungarian algorithm solves an assignment problem using a combined cost matrix $\mathbf{C}$:

$$\mathbf{C} = \alpha \left(1 - \text{IoU}\left(\hat{M},\, M\right)\right) + \beta \left(-\log\left(\text{Softmax}\left(\hat{C}_{\text{mask}}\right) + \epsilon\right)\right) \tag{11}$$

   - α and β are weighting coefficients for the IoU and classification cost components, respectively.
   - $\text{IoU}\left(\hat{M},\, M\right)$ computes the Intersection over Union between predicted and ground truth masks.
   - The negative log-likelihood term penalizes incorrect class predictions.
   - ϵ is a small constant added to avoid taking the logarithm of zero.

3. **Bounding Box Regression Loss**:

   This loss quantifies the error between the predicted bounding boxes $\hat{B}$ and the regression targets B^* (encoded ground truth boxes relative to anchor boxes A) using the Smooth L1 Loss:

$$L_{\text{bbox}} = \frac{1}{K} \sum_{j=1}^{K} \text{SmoothL1}\left(\hat{B}_j,\, B_j^*\right) \tag{12}$$

   - K is the number of anchor boxes.
   - The regression targets B^* are obtained by encoding the matched ground truth boxes B with respect to the anchors A:

$$B^* = \text{encode}\left(B,\, A\right) \tag{13}$$

   - Anchors are matched to ground truth boxes using an IoU-based matching strategy.

4. **Classification Loss**:

   The classification loss evaluates the discrepancy between the combined class predictions $\hat{C}$ and the ground truth class labels C. The predictions from both the bounding box branch $\hat{C}_{\text{bbox}}$ and the mask branch $\hat{C}_{\text{mask}}$ are combined by taking the element-wise maximum of their softmax probabilities:

$$\hat{C} = \max\left(\text{Softmax}\left(\hat{C}_{\text{bbox}}\right),\, \text{Softmax}\left(\hat{C}_{\text{mask}}\right)\right) \tag{14}$$

   The classification loss is then computed using the Cross-Entropy Loss:

$$L_{\text{class}} = \frac{1}{N} \sum_{i=1}^{N} \text{CrossEntropy}\left(\hat{C}_i,\, C_i\right) \tag{15}$$

   - N is the number of instances.
   - C_i is the ground truth class label for instance i.

This multi-task learning approach leverages the shared representations between tasks, leading to better generalization and performance compared to training each task separately. It allows the model to learn more robust and comprehensive features from the data, as the different tasks provide complementary information that enriches the learning process. By leveraging both mask predictions and bounding box regressions, the model maximizes the use of all available data from small objects. Specifically, this combined approach addresses the challenges posed by small objects, which often provide limited information due to their size. The masks capture fine-grained spatial details that might be overlooked in bounding box predictions, enhancing the accuracy of segmentation for small objects. Meanwhile, the bounding boxes provide coarse localization that can guide the mask predictions, improving robustness in detecting small objects. This synergy allows the model to effectively detect and segment small objects, enhancing both accuracy and robustness in scenarios where small object recognition is critical.

The table below, sorted by the number of parameters, provides a comparative look at various popular object detection models that we have presented in the section above. This table highlights models ranging from YOLOv3, known for its balance of speed and accuracy, to more complex architectures like Mask R-CNN, which provides high precision at the cost of increased computational resources. Notably, the Extended Masked-Attention Mask Transformer shows how recent advancements aim to reduce parameter counts while potentially enhancing model capabilities. This spectrum illustrates the trade-offs between model complexity and operational efficiency, guiding decisions in model selection based on specific application needs.

Model	**Number of Parameters (millions)**
YOLO (You Only Look Once)	62
Extended Masked-Attention Mask Transformer	76.6
Single Shot Multibox Detector (SSD)	Typically around 100
Fast R-CNN	Typically around 150
Faster R-CNN	Typically around 200
FPN with Basic Detector	Typically around 200
Mask R-CNN	Typically around 250
R-CNN	Varies, typically less than 250

Table 3.1: Comparison of Object Detection Models Sorted by Number of Parameters

4 Experiments

In the experimental phase, we selected datasets that have previously been utilized in well-recognized competitions. This allows us to benchmark the performance of our model against established baselines and leading-edge results within the field, helping us make meaningful comparisons with our model. Each dataset employed in our experiments represents a common scenario in the field of object detection, encompassing diverse environments and perspective urban life captured through ground-level cameras, aerial views provided by drones, and wide-ranging perspectives from satellites.

4.1 Datasets

For the benchmarking of our model we decided to use three very well known datasets, and we are going to analyze in this section.

4.1.1 Microsoft Common Object in COntext (MS COCO)

The Microsoft Common Objects in Context (MS COCO) dataset 2017 is a comprehensive image dataset designed for object detection, segmentation, and captioning tasks. It is known for its complexity of its images, which are primarily sourced from everyday scenes. The dataset includes 80 object categories, providing a wide range of common objects for robust training and evaluation of machine learning models. These categories encompass various items from person, bicycle, car, to stop sign and smaller objects like toothbrush.

In terms of its composition, the MS COCO 2017 dataset contains over 118,000 training images, 5,000 validation images, and a test set of around 41,000 images, bringing its total to approximately 164,000 images. This dataset is also accompanied by over 1.5 million object instances, each annotated. The annotations are formatted to support both object detection and instance segmentation tasks. Specifically, for object detection, each annotation includes not only the class label and a bounding box defined by the x and y coordinates of the top-left corner, width, and height, but also a detailed segmentation mask for each object instance, making it suitable for more granular segmentation tasks as well.

The dataset is divided into training, validation, and test splits to facilitate the training and fine-tuning of models in a structured manner. This split ensures that models can be trained on a large set of images and parameters can be fine-tuned on the validation set before final evaluations are performed on the test set. The use of MS COCO for competition and research has helped advance the field of computer vision by providing a challenging set of images and annotations that test the limits of both existing and novel visual recognition models.

Class Distribution in COCO 2017:

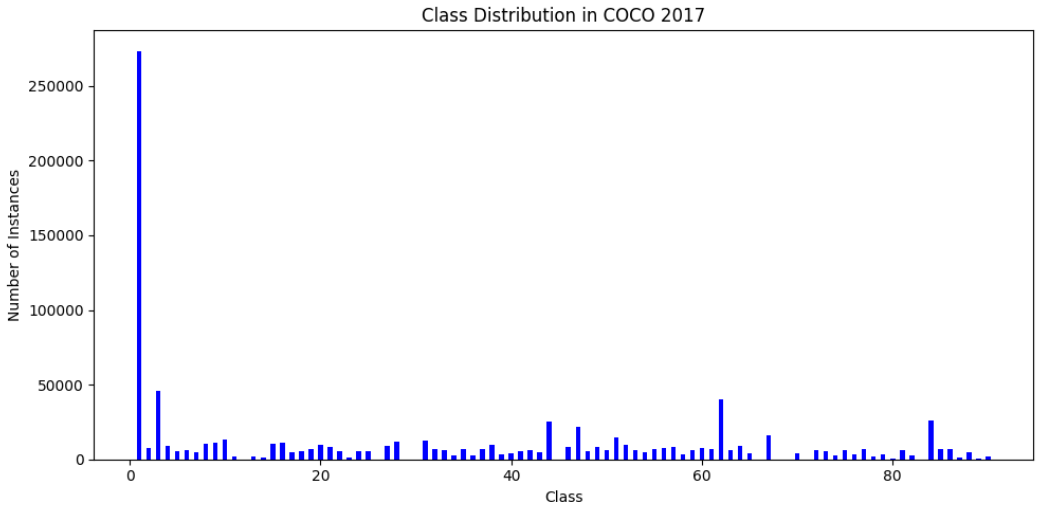

Figure 4.1: Class Distribution in COCO 2017

4.1.2 Unmanned Aerial Vehicle Small Object Detection

The UAV-SOD (Unmanned Aerial Vehicle Small Object Detection) dataset is specifically designed for advancing research in the area of object detection using aerial imagery captured by drones. This dataset focuses on the detection of small objects, which presents unique challenges due to the small scale and often complex backgrounds seen in aerial images. Here are the key details about the UAV-SOD dataset. The UAV-SOD dataset includes detailed annotations for each image, essential for supervised machine learning models such as object detectors. Annotations are provided in XML format compatible with the PASCAL VOC annotation format, which is widely used in object detection tasks. These annotations include bounding boxes that specify the coordinates of each object in the image. The objects in this dataset are annotated with their class labels, enabling not only object detection tasks but also potential use for object classification and segmentation challenges.

Class Distribution in UAV Small Object Detection:

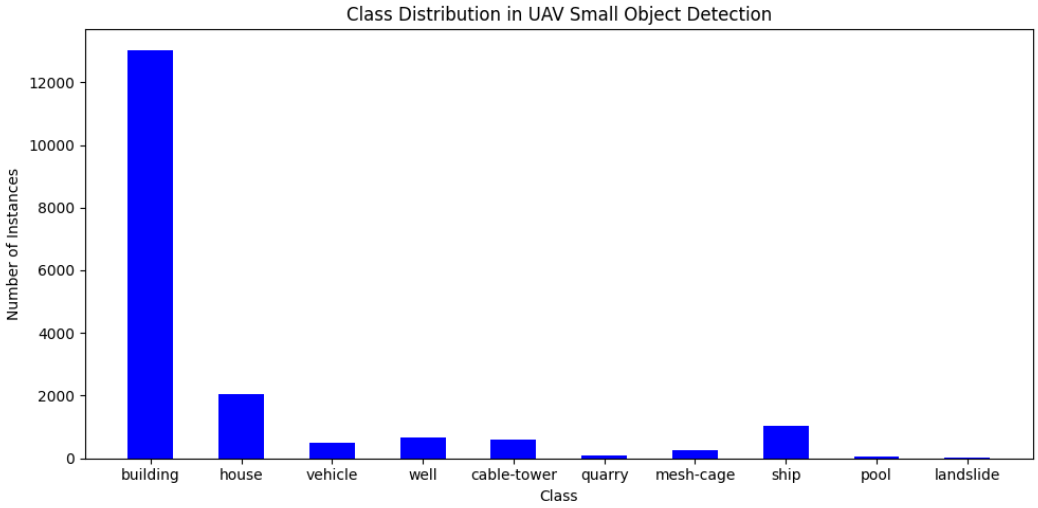

Figure 4.2: Class Distribution in UAV Small Object Detection

4.1.3 VisDrone

VisDrone is a dataset designed for drone-based object detection, featuring $10,209$ images captured across various environments using different types of drones. Each image in the dataset has a high resolution of 2000×1500 pixels. The dataset is organized into splits for comprehensive training and evaluation, consisting of $6,471$ images for training, 548 images for validation, and $3,190$ images for testing. VisDrone includes a diverse set of ten object classes, such as pedestrians, cars, vans, buses, trucks, motorcycles, bicycles, awning-tricycles, and tricycles. This wide range of categories, combined with the diverse aerial perspectives provided by drone capture, makes VisDrone an ideal resource for developing and benchmarking deep learning models focused on enhancing object detection capabilities in drone surveillance systems.

Class Distribution in VIS Drone:

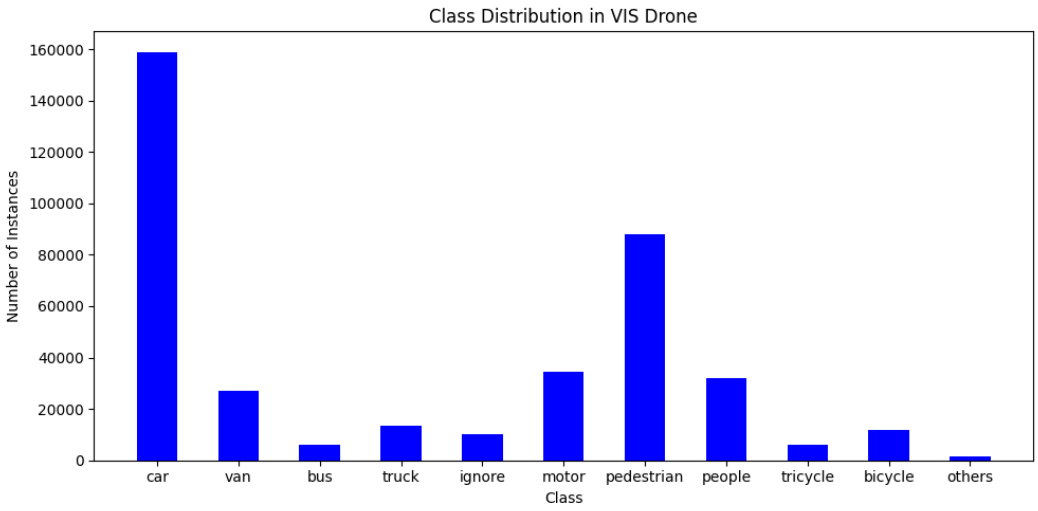

Figure 4.3: Class Distribution in VIS Drone

As we can understand from the class distribution, we can understand that class imbalance is present in the datasets. Class imbalance poses a significant challenge in training because models trained on such data may develop a bias towards more common classes and perform poorly on rare classes. We have to take this into consideration when choosing the evaluation metrics that give a more nuanced view of class-specific performance, such as per-class accuracy or the F1-score, are crucial for assessing a model's true effectiveness across the varied classes present in these datasets.

4.2 Data Pre-Processing

To ensure that the object detection models are trained on well-structured and uniform data, pre-processing steps are applied to standardize and enhance the raw dataset inputs. This process is crucial for achieving optimal model performance and is applied consistently across different datasets, such as COCO2017, UAV-SOD Drone, and Vis-Drone datasets, with specific adjustments tailored to each dataset's unique characteristics.

Firstly we have some general pre-processing steps applied across all datasets in order to make the training procedure

1. Resizing Images and Annotations: All images and corresponding annotations are resized to a uniform size of 600×600 pixels. This standardization helps maintain consistency across the dataset, which is crucial for the feature map creation with the Efficient-B7 model. The resizing function also handles the segmentation data associated with each image, ensuring that all spatial information (bounding boxes and segmentation masks) is correctly scaled and translated according to the new image dimensions.

2. Image Padding: To maintain aspect ratio without distorting the image content, padding is applied using the most common color found in the image. This approach prevents the introduction of biases that could occur if a non-representative color was used.

3. Annotation Format: The annotation files for all the datasets have a common format in order to avoid any problems with the training process. The format is the following: $x_{min}, y_{min}, x_{max}, y_{max}$, class code, $[(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)]$, where the first four values are the bounding box coordinates following the Pascal VOC format, then we have class code of the object and lastly the list of coordinates describing the mask of the object.

4. Mask Creation: The masks are a vital part of the model, since we want to use the functionality of the Masked-Attention Mask Transformer in combination with the bounding box functionality to get better and more detailed results. The datasets ViS-Drone and Unmanned Aerial Vehicle datasets do not include masks and in order to have mask data we generate masks from the bounding box data. More specifically the mask data format is the following:
$[(x_{min}, y_{min}), (x_{min}, y_{max}), (x_{max}, y_{min}), (x_{max}, y_{max})]$

5. Calculation of Mean and Standard Deviation: For normalization purposes, the mean and standard deviation of the pixel values across the dataset are computed. These statistics are essential for normalizing the images during the model training process, allowing for faster convergence and improved generalization.

6. Visualization of Data: Functions are included to plot images alongside their annotations for both training and validation sets. This visual check allows for the verification of the correct processing and annotation of the data, ensuring integrity before training begins.

Furthermore we make specific changes to each dataset in order to ensure compatibility with the model. These changes are:

1. **COCO2017 Dataset -** Annotation Conversion: The original COCO annotations are converted into a simpler text format that includes bounding box coordinates and segmentation data. This conversion facilitates the use of custom training pipelines that may not natively support COCO's complex annotation format.

2. **Vis-Drone Dataset -** Annotation Format Standardization: The annotations provided with the Vis-Drone dataset are standardized to ensure consistency with other datasets. This process includes recalculating bounding box coordinates from $[x_{min}, y_{min}, width, height]$ to $[(x_{min}, y_{min}), (x_{min}, y_{max}), (x_{max}, y_{min}), (x_{max}, y_{max})]$

3. **UAV-SOD Drone Dataset -** XML to Text Conversion: Given that UAV-SOD annotations might be provided in XML format, a conversion process is implemented to translate these XML files into a plain text format that is easier to manipulate and integrate into training workflows. The conversion also involves mapping categorical names to predefined numerical codes, which are crucial for consistent label encoding across different datasets.

These pre-processing steps are critical for preparing the data, ensuring that it meets the necessary format and quality standards required for effective model training. This comprehensive approach to data preparation helps in minimizing issues during training, leading to more robust and accurate object detection models.

4.3 Evaluation Metrics

Evaluation metrics are vital in object detection as they quantitatively measure a model's accuracy, robustness, and effectiveness in predicting correct object locations and classifications. These metrics enable developers to compare different models, optimize parameters, and ensure that the system performs well across various conditions and datasets, guiding improvements and ensuring that the deployed models meet the required performance standards. Starting with the basic definitions we have:

1. True Positives (TP) are the predictions that correctly match the class of the ground truth AND Confidence and IoU scores are higher than their thresholds.

2. False Positive (FP) are the predictions that incorrectly predict a positive class when the ground truth is actually negative OR the IoU score is lower than the threshold.

3. False Negative (FN) occurs when the model misses positive instances. It predicts a negative class when the ground truth is actually positive.

4. True Negative (TN) represents instances where the confidence score of a detection that is not supposed to detect anything is lower than the threshold. This metric is not used in the object detection field since there are too many boxes that should not detect an object in an image.

The metrics we are going to be using that are object detection specific are the Average Precision (AP), mean Average Precision (mAP), Intersection over Union (IoU), Average Recall (AR) and mean Average Recall (mAR).

Recall measures the proportion of actual positives that are correctly identified by the model, indicating the model's ability to detect all relevant instances.

$$\text{Recall} = \frac{TP}{TP + FN}$$

Precision indicates the accuracy of the positive predictions made by the model, measuring the proportion of positive identifications that were actually correct.

$$\text{Precision} = \frac{TP}{TP + FP}$$

F1-Score is the harmonic mean of precision and recall, providing a balance between the two by penalizing extreme values.

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

Intersection Over Union (IoU) assesses the overlap between predicted bounding boxes and ground truth boxes, quantifying the exactness of the location of predictions. It is a fundamental metric for determining whether a detection is a true.

$$IoU = \frac{\text{Area of Overlap}}{\text{Area of Union}}$$

Average Precision (AP) quantifies the precision of an object detector across different recall levels by taking the mean of precisions computed at the point of each of the 11 recall levels. The precision at each recall level r is interpolated by taking the maximum precision measured for a method at any recall level $\geqslant r$

$$AP = \frac{1}{11} \sum_{r \in \{0.0, 0.1, \ldots, 1.0\}} P_{\text{interp}}(r)$$

Mean Average Precision (mAP) is the average of the AP scores for all classes or over different Intersection over Union (IoU) thresholds. This metric gives an overall effectiveness of the detection model across various object categories or detection precisions, making it ideal for scenarios where multiple object types are involved.

$$mAP = \frac{1}{N} \sum_{i=1}^{N} AP_i$$

4.4 Training

To implement the Extended Masked-Attention Mask Transformer, we utilized Amazon Web Services (AWS) SageMaker for efficient training and experimentation. Specifically, we employed the *ml.p4d.24xlarge* instance type for our notebook environment, which is equipped with eight NVIDIA A100 GPUs and supports CUDA 12.1. This powerful computational setup allowed us to run the training with a batch size of four images per batch, achieving an optimal balance between convergence speed and model accuracy.

The weights of the network were initialized using the default settings provided by PyTorch, ensuring standard initialization practices. All experiments were conducted using this configuration to maintain consistency and reliability in our results. By leveraging the capabilities of AWS SageMaker and the high-performance computing resources of the *ml.p4d.24xlarge* instance, we were able to efficiently train the model and effectively utilize the limited data from small objects in images.

To provide clarity on our training setup and ensure reproducibility, the table 4.1 summarizes the key training parameters used for the Extended Masked-Attention Mask Transformer. These parameters were carefully selected based on extensive experimentation to achieve an optimal balance between convergence speed and model accuracy.

	MS COCO	**UAV-SOD**	**VisDrone**
Number of Epochs	125	75	75
Optimizer	AdamW	AdamW	AdamW
Learning Rate	1×10^{-4}	1×10^{-3}	1×10^{-3}
Batch size	4	4	4
Image size	600×600	600×600	600×600
Number of Anchors	30000	722	722

Table 4.1: Details of training parameters per dataset

4.5 Results

Following the comprehensive description of our training setup and parameters, we now present the results of our experiments with the Extended Masked-Attention Mask Transformer. This section delves into the performance metrics achieved by the model, highlighting its effectiveness in both object detection and instance segmentation tasks. We evaluate the model's ability to accurately detect and segment objects of various sizes within images, with a particular focus on its performance on small objects, which pose significant challenges due to their limited visual information.

4.5.1 Microsoft Common Object in COntext (MS COCO)

Model	**mAP(@0.5)**	**mAP(@0.5:0.95)**	**Parameters**
EMAMT	125	75	75
MAMT	125	75	75

Table 4.2: Results of training for COCO dataset

4.5.2 Unmanned Aerial Vehicle Small Object Detection

Model	**mAP(@0.5)**	**mAP(@0.5:0.95)**	**Parameters**
EMAMT	125	75	75
MAMT	125	75	75

Table 4.3: Results of training for UAV-SOD dataset

4.5.3 VisDrone

Model	**mAP(@0.5)**	**mAP(@0.5:0.95)**	**Parameters**
EMAMT	125	75	75
MAMT	125	75	75

Table 4.4: Results of training for VisDrone dataset

5 Discussion

6 Conclusion

Bibliography

A Appendix A