SCENE CLASSIFICATION AND TARGET DETECTION USING VISION TRANSFORMER ARCHITECTURE

PHASE I REPORT

Submitted by

M VIJAYA PRAKASH

(Reg. No: 2022603026)

in partial fulfillment for the requirements of the degree of

MASTER OF ENGINEERING IN AVIONICS

DIVISION OF AVIONICS

DEPARTMENT OF AEROSPACE ENGINEERING

ANNA UNIVERSITY, CHENNAI-600044

JANUARY 2024

BONAFIDE CERTIFICATE

Certified that this thesis titled **"SCENE CLASSIFICATION AND TARGET DETECTION USING VISION TRANSFORMER ARCHITECTURE"** is the bonafide work of **M VIJAYA PRAKASH (Roll No. 2022603026)** who carried out the work under my supervision, for the partial fulfilment of the requirements for the award of the degree of **MASTER OF ENGINEERING** in **AVIONICS**. Certified further that to the best of my knowledge the work reported herein does not form part of any other thesis or dissertation on the basis of which a degree or award was conferred on an earlier occasion on this or any other candidate.

Dr. K.M. PARAMMASIVAM
Professor and Head,
Department of Aerospace Engineering,
Madras Institute of Technology,
Anna University,
Chennai – 600 044.

Dr. G. ANITHA
Professor,
Division of Avionics,
Madras Institute of Technology,
Anna University,
Chennai – 600 044.

திட்டப்பணி அறிக்கை

மாணவர் பெயர்	: மா விஜய பிரகாஷ்
பதிவு எண்	: 2022603026
பட்டம் மற்றும் கிளை	: முதுநிலை பொறியியல், வானுர்தி மின்னணுவியல்
திட்டப்பணியின் தலைப்பு	: பார்வை டிரான்ஸ்பார்மர் கட்டமைப்பைப் பயன்படுத்தி காட்சி வகைப்பாடு மற்றும் பொருள் கண்டறிதல்
திட்டப்பணி நிறைவு செய்த திங்களும் ஆண்டும்	: மார்கழி , 2024
திட்டப்பணியின் மேற்பார்வையாளர்பெயர் மற்றும் பதவி	: முனைவர் க.அனிதா, பேராசிரியர், வானுர்தி மின்னணுவியல் பிரிவு, வான்வெளி பொறியியல் துறை, மெட்ராஸ் தொழில்நுட்ப கல்லூரி, சென்னை – 600044.

ஆய்வு சுருக்கம்

ட்ரோன் மோதலைத் தவிர்ப்பது, ட்ரோன் பாதுகாப்பு மற்றும் உட்புற சூழ்நிலையில் தன்னியக்க ஆளில்லா வான்வழி வாகனம் (UAV) சுயமாக தரையிறங்குதல் ஆகியவற்றில் துல்லியமான ட்ரோன் வழிசெலுத்தல் மிகவும் விரும்பப்படுகிறது. உட்புற காட்சி அங்கீகாரம் UAV இன் வழிசெலுத்தலின் முதல் படியாகும். செயல்முறையை விரைவுபடுத்த, இயற்கை மொழி செயலாக்கம் (NLP) பயன்படுத்தப்படுகிறது, ஏனெனில் இது வெளிப்படையாக நிரலாக்கம் இல்லாமல் எடுத்துக்காட்டுகளிலிருந்து கற்றுக்கொள்ள முடியும். விஷன் டிரான்ஸ்ஃபார்மர் (ViT) என்பது மின்மாற்றி கட்டமைப்பில் உருவாகி வரும் ஒரு துறையாகும், குறிப்பாக அதன் பட செயலாக்கம் மற்றும் படத்தை அடையாளம் காண அறியப்படுகிறது, மேலும் இந்த முக்கியமான பணியானது உட்புற படிக்கட்டு மற்றும் ட்ரோன் தரவுத்தொகுப்பைப் பயன்படுத்தி இந்தத் திட்டத்தில் மறுமதிப்பீடு செய்யப்படுகிறது. நாங்கள் பல்வேறு CNN மற்றும் ViT-சார்ந்த மாதிரிகளை உருவாக்குகிறோம், ட்ரோன் கண்டறிதலுக்கு, ஒரு அடிப்படை ViT ஆனது எங்களின் சிறந்த CNN அடிப்படையிலான பரிமாற்ற கற்றல் மாதிரிகளை விட வலுவான செயல்திறனை அடைய முடியும் என்பதை நிரூபிக்கிறது. படிக்கட்டு (காட்சி) கண்டறிதல் மற்றும் UAV (பொருள்) கண்டறிதலுக்கான அதிநவீன YOLO மற்றும் சோதனை ViT அடிப்படையிலான YOLOS ஆகியவற்றை நாங்கள் செயல்படுத்தியுள்ளோம். அதே சகாப்தத்தில் CNN ஐ விட ViT சிறப்பாக செயல்படுவதை நாங்கள் காண்கிறோம், ஆனால் அதிநவீன CNN டிடெக்டர்களின் திறன்களை முழுமையாக மிஞ்ச அதிக பயிற்சி தரவு மற்றும் கணக்கீட்டு சக்தி, செயல்திறன் சார்ந்த வடிவமைப்புகள் தேவை.

ABSTRACT

Accurate drone navigation is strongly desired in drone collision avoidance, drone defense and autonomous Unmanned Aerial Vehicle (UAV) self-landing in indoor scenario. Indoor scene recognition is the first step in the navigation of UAV. To make the process faster, Natural Language Processing (NLP) is applied because it can learn from examples without explicitly programming. Vision Transformer (ViT) is an emerging field in transformer architecture especially known for its image processing and image identification. With the recent emergence of the Vision Transformer (ViT), this critical task is reassessed in this project using an indoor staircase, corridor and drone dataset. We construct various CNN and ViT-based models, demonstrating that for drone detection, a basic ViT can achieve performance more robust than our best CNN-based transfer learning models. We have implemented the state-of the-art You Only Look Once and the experimental ViT-based You Only Look At One Sequence for staircase (scene) detection and UAV (object) detection. We find that ViT outperforms CNN at the same epoch, but also requires more training data, computational power, and sophisticated, performance-oriented designs to fully surpass the capabilities of cutting-edge CNN detectors. We summarize the distinct characteristics of ViT and CNN models to aid future researchers in developing more efficient deep learning models.

ACKNOWLEDGEMENT

I would like to thank **Dr. J. PRAKASH**, Dean, Madras Institute of Technology, for providing the opportunity to carry out the project work successfully.

I express my gratitude to **Dr. K.M. PARAMMASIVAM**, Professor and Head, Department of Aerospace Engineering, Madras Institute of Technology, for providing all the facilities that were needed for the proper execution of this project.

I wish to express my sincere gratitude and thankful to my Project Guide **Dr. G. ANITHA**, Professor, Division of Avionics, Department of Aerospace Engineering, Anna University, MIT Campus, Chennai for her valuable guidance and timely support she rendered during the course of this project.

It is worth mentioning my friends and colleagues from the aerospace department for extending their kind help whenever the necessity was in demand. I thank one and all who have directly/indirectly helped me in making this project a great success.

I would also like to extend my regards to our faculty members for their constant support during the coursework.

Also, the countless researchers and scientific personnel who shared their work in the various forms of publications in internet deserve a huge respect and a note of gratitude, without whom, it would have been much more difficult to grasp the knowledge required for this project.

M VIJAYA PRAKASH

(2022603026)

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 OBJECTIVE

The objective of scene classification using a vision transformer model with self-attention mechanism is to recognize target object as real-time mission objective. Self-attention allows the model to consider the dependencies between all pairs of input tokens (patches, in the case of ViT) in each sequence simultaneously. This enables ViT to capture long-range dependencies and relationships between image regions more effectively than traditional convolutional neural networks (CNNs). We propose an implementation of a Vision Transformer based model, which collects the target object data and perform preprocessing for self-attention mechanism, then trained with relative to the pre-trained model such as ImageNet that handles time-varying dynamic real time implementation.

1.2 OBJECT RECOGNITION

Object recognition is a computer vision task that involves identifying and locating objects of interest within an image or a video frame. The goal is to draw bounding boxes around the detected objects and assign class labels to each box. It involves categorizing the detected objects into predefined classes or categories. Each bounding box is associated with a class label indicating the type of object it represents. In Image classification, it takes an image as an input and outputs the classification label of that image with some metric (probability, loss, accuracy, etc.). The art of object detection could be executed through multiple methods such as Region-Based CNNs (R-CNNs), You Only Look Once (YOLO), single shot multibox detector (SSD), RetinaNet, CenterNet, EfficientDet and many more.

1.3 SCENE CLASSIFICATION

Scene recognition is a navigation technology related to computer vision and image processing that deals with the identification of different scenes like classroom, corridor, staircase, test field, etc. The topic of scene classification has been an active research field lately to face the challenging problem of effectively interpreting remote-sensing images. It is the task of taking an image and correctly labeling it to a predefined class. Scene classification is an important task for many applications, such as indoor navigation, obstacle avoidance, land management, mapping and localization. The main difference between object recognition and scene recognition is that object recognition focuses on classifying prominent objects in the foreground, whereas scene recognition uses the layout of objects within the scene, in addition to the ambient context, for identifying the scenes.

The early works on scene classification were based on handcrafted features, manually extracted by humans, including local binary patterns (LBP), histogram of oriented gradients (HOG), and the scale-invariant feature transform (SIFT). Conventional scene-classification methods depend on encoding handcrafted features with different models such as the bag-of-words, Fisher vectors, or the vector of locally aggregated descriptors (VLAD).

1.4 NEED FOR SCENE CLASSIFICATION

New age intelligent systems are expected to perform with different situations without human intervention. It is important for them to know the place or context, which enhances their understanding of past events to predict future. An autonomous UAV can be considered as one example of such an intelligence system. Classification of a scene as being indoors or outdoors is a challenging task in the navigation of these autonomous

UAV. Better indoor versus outdoor scene categorization will help UAV navigation where global positioning systems (GPS) signals are not available. Interruption of GPS signals by dense and tall buildings, by trees and inside buildings occurs in both the indoor and outdoor environment, and to overcome this limitation, the UAV needs to have the ability to recognize the difference.

The essential goal for scene recognition is to assign the semantic labels to the given images, these semantic labels are defined by human beings including different natural views, indoor scenes, outdoor environments etc. However, scene recognition not only concerns the existence of objects but also the semantic relations between objects and the contextual information with respect to the background.

The main difference between object recognition and scene recognition is that object recognition focuses on classifying prominent objects in the foreground, whereas scene recognition uses the layout of objects within the scene, in addition to the ambient context, for identifying the scenes.

1.5 LITERATURE REVIEW

The literature survey is done to understand the current solutions being approached to the problem statement and the issues being encountered by them are noted below. The shortcomings faced by each work are being considered as an input to this project work.

1. **Dosovitskiy, Alexey, et al. "An image is worth 16x16 words: Transformers for image recognition at scale. arXiv 2020."** ***arXiv preprint arXiv:2010.11929*** **(2010).**

This paper explains the advantages of vision transformers over traditional convolutional neural networks. It states that in vision, attention is either applied in

conjunction with convolutional networks, or used to replace certain components of convolutional networks while keeping their overall structure in place. Unlike prior works using self-attention in computer vision, the author does not introduce image-specific inductive biases into the architecture apart from the initial patch extraction step. Instead, they interpret an image as a sequence of patches and process it by a standard Transformer encoder as used in NLP. This method performed well when coupled with pre-training on large datasets.

2. **Gehrig, Mathias, and Davide Scaramuzza. "Recurrent vision transformers for object detection with event cameras."** ***Proceedings of the IEEE on Computer Vision and Pattern Recognition*****. 2023.**

Our research highlights that recurrent vision transformers can be trained from scratch to reach state-of-the-art performance in object detection. The ViT architecture consists of a stage design that is repeatedly applied to create a multistage hierarchical neural network. Each stage compactly incorporates convolutional priors, local- and sparse global attention and recurrent feature aggregation. Their approach only uses event streams to detect objects.

3. **Khan, Salman, et al. "Transformers in vision: A survey."** ***ACM computing surveys (CSUR)*** **54.10s (2022): 1-41.**

The survey aims to provide a comprehensive overview of the Transformer models in the computer vision discipline. It starts with an introduction to fundamental concepts behind the success of Transformers and then cover extensive applications of transformers in vision including popular recognition tasks (e.g., image classification, object detection, action recognition, and segmentation). In the end it systematically

highlights the key strengths and limitations of the existing methods and particularly elaborates on the important future research directions.

4. **Vaswani, Ashish, et al. "Attention is all you need."** ***Advances in neural information processing systems*** **30 (2017).**

The paper proposes a new simple network architecture, the Transformer, based solely on attention mechanisms, dispensing with recurrence and convolutions entirely. This was executed by replacing the recurrent layers most used in encoder-decoder architectures with multi-headed self-attention. The paper played a major role in explaining the advantages of NLP over RNN architecture.

5. **Bello, Irwan, et al. "Attention augmented convolutional networks."** ***Proceedings of the IEEE/CVF international conference on computer vision*****. 2019.**

This research played a significant role in computer vision as it enhanced CNNs by replacing some convolutional layers with self-attention layers, which led to improvement in 2D image classification for the first time. It declares that Attention Augmentation leads to consistent improvements in image classification on ImageNet and object detection on COCO across many different models and scales, including ResNets while keeping the number of parameters similar. However, this method faced high computational cost because the large size of the image causes an enormous growth in the time complexity of self-attention.

6. **Ramachandran, Prajit, et al. "Stand-alone self-attention in vision models."** ***Advances in neural information processing systems*** **32 (2019).**

The paper focuses on implementing attention model as a standalone primitive for vision instead of simple augmentation on top of convolutional layers. They set up a fully attention model by replacing all convolutional layers with self-attention layers. The experiment was carried out in ImageNet classifications and COCO object detection tasks. The significant output recorded was that the training efficiency and computational demand of an attention-based architecture is favorable to a traditional convolution, the resulting network is slower in wall-clock time. The reason for this record is the lack of optimized kernels available on various hardware accelerators.

1.5 RESEARCH GAP AND EXECUTION

Until today, Vision Transformer has often only been used to solve individual photos, but never to provide a time-varying dynamic real time image processing. Investigating the transfer learning capabilities of Vision Transformers and exploring whether pre-trained ViT models on datasets can be effectively fine-tuned for indoor navigation tasks, considering the domain gap between indoor and outdoor environments and object recognition. Plan of execution was to develop ViT model for indoor location and target object recognition by collecting target dataset. It vastly simplifies the tuning of the training, since the tuning parameters are costs that directly related to the accuracy of the training model. To achieve this, we leverage the self-attention model to implement an independent self-learning vision transformer model.

CHAPTER 2

VISION TRANSFORMERS

2.1 EVOLUTION OF TRANSFORMERS IN COMPUTER VISION

Besides CNNs, a new type of deep-learning model called Transformers has been proposed and received some popularity in computer vision. Transformers rely on a simple but powerful procedure called attention, which focuses on certain parts of the input to get more efficient results. Currently, they are considered state-of-the-art models in sequential data, in particular natural language processing (NLP) methods such as machine translation, language modeling, and speech recognition. The architecture of the Transformer developed by Vaswani et al. is based on the encoder–decoder model, which transforms a given sequence of elements into another sequence. The main motivation for transformers was to enable parallel processing of the words in a sentence, which was not possible in RNNs because they take words of a sentence one by one. Inspired by the success of Transformers in NLP, new research tries to apply Transformers directly to images. This is a challenging task, due to the need for self-attention application that every pixel attends to all other pixels. For images, this is very costly because the image contains a huge number of pixels. Researchers tried several approaches to apply Transformers to images. Some works combined CNN architectures with self-attention. For example, Bello et al. enhanced CNNs by replacing some convolutional layers with self-attention layers, which led to improvement in image classification. Wang et al. proposed a method to generate powerful features by selectively focusing on critical parts or locations of the image, then processing them sequentially. Wu et al. used the Transformer on top of the CNN; first they extracted feature maps using a CNN, then fed them to stacked visual Transformers to process visual tokens and compute the output. Ramachandran et al. first started to use self-attention as a stand-alone building block for

vision tasks instead of a simple augmentation on top of convolutional layers. They set up a fully attention model by replacing all convolutional layers with self-attention layers. Chen et al. proposed a method that applies Transformers to raw images with reduced resolution and reshaped into text like long sequences of pixels. In a very recent contribution, and different from previous works, Dosovitskiy et al. applied a standard Transformer directly to images by splitting the image into patches not focusing on pixels, then input to the Transformer the sequence of embeddings for those patches. The image patches were treated as tokens in NLP applications. These models led to very competitive results on the ImageNet dataset. In this work, we will exploit these pretrained models for transferring knowledge to the case of remote-sensing imagery. Indeed, to the best of our knowledge in remote-sensing scene-classification tasks, convolutional architectures remain dominant, and Transformers have not yet been widely used as the model choice in classification. For instance, He et al. proposed a model derived from the bidirectional encoder representations called BERT used in the natural language processing field to hyperspectral images. The method is based on several multi-head self-attention layers. Each head encodes the semantic context-aware representation to obtain discriminative features that are needed for accurate pixel-level classification.

2.2 VISION TRANSFORMERS (VIT)

A vision transformer (ViT) is a neural network that is used for computer vision tasks. It is a type of transformer used for visual tasks like image processing. Vision transformer breaks down an input image into a series of patches, then it serializes each patch into a vector and finally it maps it to a smaller dimension with a single matrix multiplication. It was first proposed and executed successfully by Dosovitskiy, Alexey, et al. "An image is worth 16x16 words: Transformers for image recognition at scale. arXiv 2020." by training a transformer encoder on ImageNet, attaining very good results

compared to familiar convolutional architectures. The ViT is a visual model based on the architecture of a transformer originally designed for text-based tasks. The ViT model represents an input image as a series of image patches, like the series of word embeddings used when using transformers to text, and directly predicts class labels for the image. ViT exhibits an extraordinary performance when trained on enough data, breaking the performance of a similar CNN.

Vision Transformer (ViT) is the first work to showcase how transformers can 'altogether' replace standard convolutions in deep neural networks on large-scale computer vision datasets. They applied the original transformer model (with minimal changes compared to the version used for NLP tasks) on a sequence of image 'patches'. The transformer model was pre-trained on a large propriety dataset of images collected by Google and later fine-tuned to downstream recognition benchmarks e.g., ImageNet. This is an important step since pre-training on a medium range dataset would not give results with a ViT. This is because CNNs encode prior knowledge about the image domain that reduces the need of data as compared to transformers which must discover such knowledge rules from very large-scale datasets. The main architecture of the model is very similar to language transformers. Instead of a 1D sequence of language embeddings, 2D images patches are flattened in a vector form and fed to the transformer as a sequence. These vectorized patches are then projected to a patch embedding using a linear layer and position embedding is attached with it to encode location information. Importantly, a token is appended at the beginning of the input to the transformer. The output representation corresponding to the first position is then used as the global image representation for the image classification task.

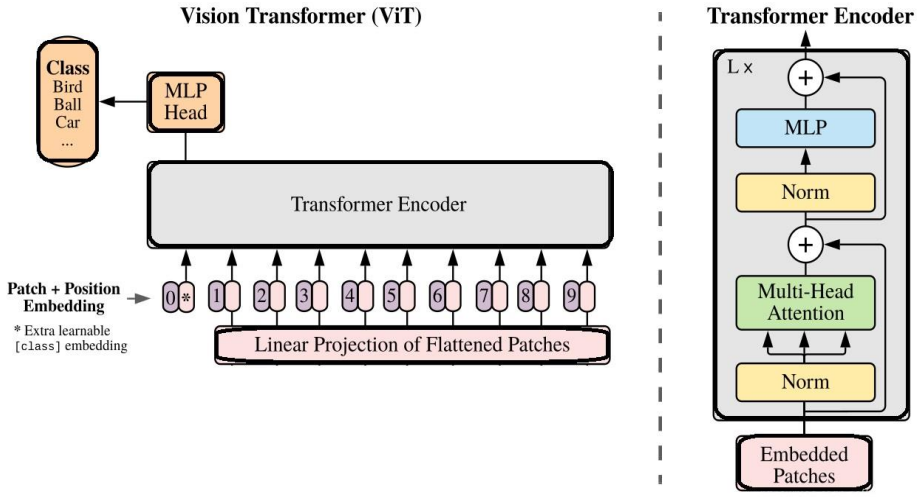


Figure 2.1 - ViT model overview (The illustration of the Transformer encoder was inspired by Vaswani et al. (2017))

2.3 VISION TRANSFORMER VARIANTS

To experiment on the effect of increasing the model size on the classification accuracy, different versions of Vision Transformer have been proposed: the "ViT-Base", the "ViT-Large", and the "ViT-Huge". The three versions differ in the number of the encoder's layers, the hidden dimension size, the number of attention heads used by MSA layer, and the MLP classifier size. Each one of these models is trained with a patch of size 16 × 16 and 32 × 32. The "ViT-Base" model has 12 layers in the encoder, with hidden size 768, and uses 12 heads in the attention layer. The other version uses larger numbers; the "ViT-Large" for example, has 24 layers, 16 attention heads, and a hidden dimension of size 1024. The "ViT-Huge" has 32 layers, 16 attention heads, and a hidden size of 1280.

Model	Number of layers	Hidden size D	MPL Size	Heads	Number of parameter
ViT - Base	12	768	3072	12	86 M
ViT - Large	24	1024	4096	16	307 M
ViT - Huge	32	1280	5120	16	632 M

Table 2.1 - Comparative summary of the transformer versions

The experimental results on Vision Transformers of different sizes have shown that using relatively deeper models is important to get higher accuracy. Moreover, choosing a small patch dimension increases the sequence length n, which in turn improves the overall accuracy of the model. Another important finding is that attention heads at the earlier layers of the Vision Transformer can attend image regions at high distances. This ability increases as the depth of the model increases. This is different from the CNNs-based models, in which earlier layers can only detect local information and global information can only be detected at the higher layers of the network.

2.4 EFFECTIVE PATCHING METHOD

The general vision transformer suffers from two weaknesses. Firstly, it requires much longer training time than conventional CNN based detectors. Secondly, it achieves low detection performance on small objects. The weaknesses come from the patching and sequencing method that the images are divided into patches which do not contain useful location information. In our method, we address this by introducing a new and effective patching method. In short, we use overlapping patches. A single path shares m pixels with its neighbor patches. By changing the number m, it can share more information with the neighbors. This can also help to attend to small sampling locations in the feature maps.

2.5 ViT BACKBONE

Vision Transformer (Dosovitskiy et al., 2021) has proved that a transformer architecture can also be used in computer vision tasks besides the general natural language processing tasks. It achieves this by taking images as sequences of image patches. Even though the performance is not state of-the-art yet, it opens a door to connect computer vision methods with natural language processing methods. The Vision Transformer backbone. The vision transformer first reshapes the image of size $H \times W \times C$ into a sequence of flattened 2D image patches of shape $N \times P \times P \times C$. H is the image height. W is the image width. C is the image channel. P is the patch size and N is the number of patches, which is also the input length. Then a linear layer is used to flatten the image patches into vectors of dimension D. Then a general transformer encoder is used to map the vectors to image representations z. Like BERT (Devlin et al., 2019), the vision transformer also used a learnable embedding to the sequence of the embedded patches, which corresponds to a classification token. It also uses positional encoding to help the transformer remember the input sequence orders. The Transformer encoder (Vaswani et al., 2017; Yang et al., 2019) consists of several multiheaded self-attention layers. Layer normalization is applied before every block, and residual connections after every block.

Because the two-dimensional neighborhood structure is used sparsely, and to increase the inductive bias of the transformer model, (Dosovitskiy et al., 2021) explores how to increase the inductive bias and different hybrid model architectures, where the input sequence can be formed from feature maps of a CNN. A special case is that the patches can have spatial size 1×1, which means that the input sequence is obtained by simply flattening the spatial dimensions of the feature map and projecting to the Transformer dimension. For more details of the model, please refer to (Dosovitskiy et al., 2021).

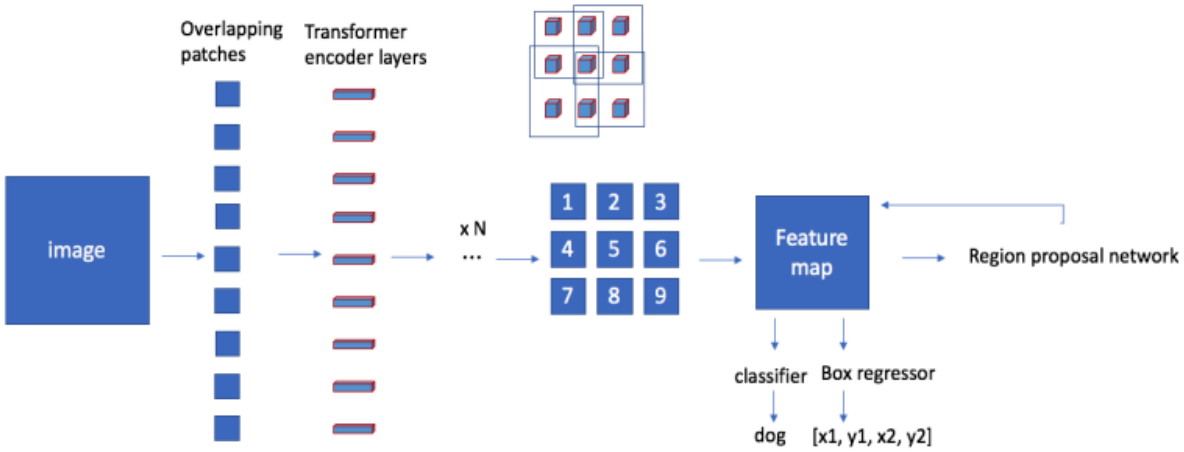


Figure 2.2 - Model Architecture

2.6 SELF-ATTENTION OVER IMAGES

Given an input tensor of shape (H, W, F_{in}), we flatten it to a matrix $X \in R^{HW \times Fin}$ and perform multi-head attention as proposed in the Transformer architecture. The output of the self-attention mechanism for a single head h can be formulated as:

$$O_h = \text{Softmax}\left((XWq)(XWk)/(d^h_k)^{1/2}\right)^T * (XW_v) \qquad \text{---[1]}$$

where $W_q, W_k \in R\ F^{in \times d}$ and $W_v \in R\ F^{in \times d}$ are learned linear transformations that map the input X to queries $Q = XW_q$, keys $K = XW_k$ and values $V = XW_v$. The outputs of all heads are then concatenated and projected again as follows:

$$MHA(X) = \text{Concat}[O_1,...,O_{Nhi}]*W^O \qquad \text{---[2]}$$

where $WO \in R^{dv \times dv}$ is a learned linear transformation. MHA(X) is then reshaped into a tensor shape (H, W, d_v) to match the original spatial dimensions. We note that multi-head attention incurs a complexity of $O(HW)^2 d_k)$ and a memory cost of $O((HW)^2 N_h)$ as it required to store attention maps for each head.

2.7 TWO-DIMENSIONAL POSITIONAL ENCODINGS

Without explicit information about positions, self-attention is permutation equivariant:

$$MHA(\pi(X)) = \pi(MHA(X)) \quad \text{---[3]}$$

for any permutation π of the pixel locations, making it ineffective for modeling highly structured data such as images. Multiple positional encodings that augment activation maps with explicit spatial information have been proposed to alleviate related issues.

Relative positional encodings: Introduced for the purpose of language modeling, relative self-attention augments self-attention with relative position encodings and enables translation equivariance while preventing permutation equivariance. We implement two-dimensional relative self-attention by independently adding relative height information and relative width information. The attention logit for how much pixel $i = (i_x, i_y)$ attends to pixel $j = (j_x, j_y)$ is computed as:

$$l_{i,j} = q_i^T * (d^h_k (k_j + r^W_{jx-ix} + r^H_{jy-iy}) \quad \text{---[4]}$$

where q_i is the query vector for pixel i (the i-th row of Q), kj is the key vector for pixel j (the j-th row of K) and r^W_{jx-ix} and r^H_{jy-iy} are learned embeddings for relative width $j_x - i_x$ and relative height $j_y - i_y$, respectively. The output of head h now becomes:

$$O_h = \text{Softmax}\left((QK^T + S^{rel}_H + S^{rel}_W) / (d^h_k)^{1/2} \right) * V \quad \text{---[5]}$$

where S^{rel}_H, $S^{rel}_W \in R^{HW \times HW}$ are matrices of relative position logits along height and width dimensions that satisfy $S^{rel}_H[i, j] = q^T_i r^H_{jy-iy}$ and $S^{rel}_W[i, j] = q^T_i r^W_{jx-ix}$. As we consider relative height and width information separately, $S^{rel}_{H \text{ and}} S^{rel}_W$ also satisfy the properties $S^{rel}_W[i, j] = S^{rel}_W[i, j + W]$ and $S^{rel}_H[i, j] = S^{rel}_H[i + H, j]$, which prevents from having to compute the logits for all (i, j) pair. The relative attention algorithm explicitly stores all relative embeddings r_{ij} in a tensor of shape (HW, HW, d^h_k), thus incurring an additional memory cost of $O((HW)^2 d^h_k)$. This compares to $O((HW)^2 N_h)$

for the position-unaware version self-attention that does not use position encodings. As we typically have $Nh < d^h_k$, such an implementation can prove extremely prohibitive and restrict the number of images that can fit in a minibatch. Instead, we extend the memory efficient relative masked attention algorithm to unmasked relative self-attention over 2 dimensional inputs. Our implementation has a memory cost of $O(HW d^h_k)$. The relative positional embeddings r^H and r^W are learned and shared across heads but not layers. For each layer, we add $(2(H + W) - 2)*d^h_k$ parameters to model relative distances along height and width.

CHAPTER 3

INDOOR SCENE CLASSIFICATION

The working environment of stair detection determines that a related algorithm usually runs on some small, embedded devices. This requires an extremely low computational cost to achieve better real-time performance. To solve this problem, the method used by most algorithms is reducing the input data, namely, reducing the size of the input image or the number of three-dimensional point clouds.

The working nature of stair detection determines that a related algorithm should have high reliability and accuracy. The most challenging problem with stair detection is the adaptability of an algorithm to deal with extreme lighting conditions, special structures, severe occlusions and special materials. For the method based online extraction, these problems will be fatal. The reason for this is that the limitations of traditional computer vision based artificial feature extraction make the algorithms difficult to adapt to complex and changeable scenarios. For the method based on plane extraction, the acquisition of point clouds depends on light detection and ranging (LiDAR) or depth cameras, which are not affected by stair texture features and lighting conditions. However, LiDAR and binocular sensors are often expensive and still cannot solve the problem of severe occlusion. In addition, when detecting objects with texture features and structures that are like stairs, these objects are often misidentified as stairs. The reason for this is that an algorithm based on feature extraction cannot obtain high-level semantic information about stairs like humans, which often leads to false detection and missed detection.

With the above motivations, the proposed end-to-end method based on NLP has an extremely fast speed and solves the problem regarding adaptability to different scenarios with monocular vision. Since ResNet established the dominant position in computer vision in 2012, NLP have rapidly developed in various fields of computer vision due to their strong learning abilities and unique perception modes. The reason for

introducing ViT into stair detection is that an artificial neural network can learn the texture features and high-level semantic information of stairs simultaneously and obtain better robustness by learning dataset that contain various detection scenarios.

3.1 DATASET [1]

The IITM-SCID2 Dataset is a challenging benchmark dataset containing 902 images of indoor and outdoor scenes. From this dataset, 193 indoor images and 200 outdoor images were used in the training phase, to train the SVM classifier, while in the testing phase, 249 indoor images and 260 outdoor images were used, based on the trained scene recognition model. The Indoor-Outdoor Dataset consists of eight outdoor scene categories, including tall buildings, cityscapes, streets, highways, mountains, forests, open country, and coastal images, in a 4485-image dataset, and three indoor scene categories, including corridors, staircases, and rooms. Real time image frames were acquired from MAVs, with an image resolution of 1280 × 720 pixels. The dataset contained 1100 training images (300 indoor and 800 outdoor images), and 550 testing images (150 indoor and 400 outdoor images). The MIT-67 Dataset contains 15,620 images in 67 categories of complex indoor scenes.

Figure 3.1 - Training Dataset

3.2 DATA PRE-PROCESSING

Image pre-processing is also called image restoration. It involves the degradation and noise during the processing of image. It produces a corrected image that is close as possible characteristics of the original image. The first step is the pre-processing, which is to resize the input images. My dataset consists of varying dimensional images, we must resize the image to 256*256 because it reduces the computational complexity along with computational time. The input layers contain image data. Image data is represented by three-dimensional matrix, where the height and width of that matrix correspond to the dimensions of the image, and the third dimension corresponds to the image channels, for example, if image is colored, the number of channels will be 3 (RGB) representing the colors Red, Green, Blue. So, the input color images are converted into three-dimensional matrix of size (256,256,3) to match the input node count of the model.

3.3 BATCH SIZE

The batch size is several samples processed before the model is updated. In ViT and many other deep learning models, training is typically done using mini batches, where a subset of the entire training dataset is processed at once. The choice of batch size depends on various factors, including the available hardware resources and the dataset size. Larger batch sizes can lead to faster training times, as more samples are processed in parallel, taking advantage of parallelization on GPUs. However, larger batch sizes also require more GPU memory. Smaller batch sizes may be used when working with limited GPU memory or when training on smaller datasets. Smaller batch sizes may introduce more noise in the parameter updates, potentially leading to a model that generalizes better, but it may also slow down the training process.

3.4 EPOCH

The appropriate number of epochs depends on various factors, including the complexity of the task, the size of the dataset, and the learning rate. Too few epochs might result in the model not learning the underlying patterns in the data, leading to underfitting. On the other hand, too many epochs can lead to overfitting, where the model becomes overly specialized to the training data and performs poorly on new, unseen data. When using Vision Transformers or any deep learning model, the choice of the number of epochs is part of the hyperparameter tuning process, and it often involves experimentation to find the best trade-off between training long enough to learn useful representations and avoiding overfitting.

3.5 EVALUATING METRICS

		ACTUAL	
		A	B
PREDICTED	A	True A (TP)	False A, Actually B (FP)
	B	False B, Actually A (FN)	True B (FP)

Table 3.1- Confusion Matrix for a Multi-categorical Classification Model

For predictions, there are four important terms: - True Positive (TP), True Negative (TN), False Positive (FP) and False Negative (FN). TP and TN represent the cases when the model correctly predicts corridor and staircase correctly, whereas FP and FN are the cases when the opposite results are obtained. The Precision metric shows what percent of predictions are correct. Recall describes what percent of positives are correctly identified. The F-beta score is the percentage of positive predictions that are correct.

$$Accuracy = \frac{Number\ of\ correct\ predictions\ (TP+TN)}{Total\ Number\ of\ predictions\ (TP+TN+FP+FN)}$$

$$Precision = \frac{TP}{TP + FP} \qquad Recall = \frac{TP}{TP + FN}$$

$$Fbeta_{Measure} = \frac{(1 + beta^2) * Precision * Recall}{beta^2 * Precision + Recall}$$

3.6 RESULTS

The Model is trained on training dataset of 350 images for 50 epochs and validated using validation set of 120 images.

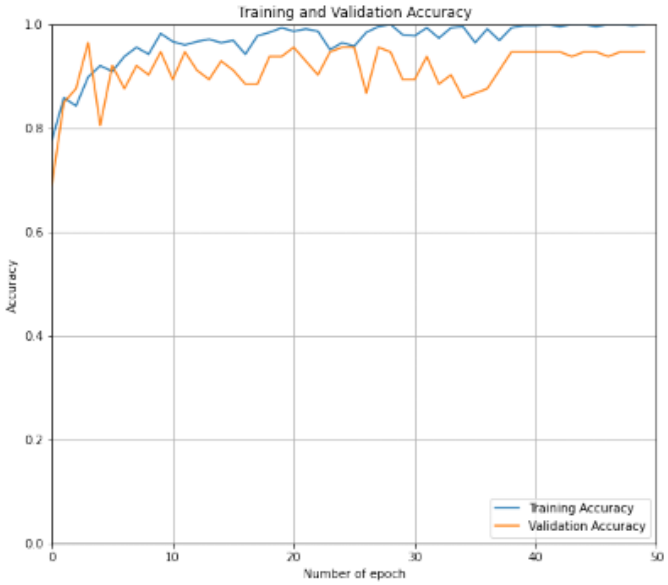


Figure 3.2 – Number of epoch vs Accuracy

The accuracy and loss function for both training and validation set is plotted for the number of epochs. The model has converged after 40 epochs. Since the training accuracy had not increased further after that.

	Precision	Recall	F1 Score	F2 Score	F0.5 score
Corridor	0.96	0.923	0.941	0.93	0.953
Staircase	0.92	0.958	0.939	0.95	0.927

Table 3.2 – Model score

Saliency maps are visualization tools used to understand which parts of an input image are most influential in making predictions by a neural network. They highlight regions of the input that the model focuses on when making its decision. It aims at resizing an image by expanding or shrinking the noninformative regions. For Vision Transformers (ViT), which are a type of neural network architecture commonly used for image classification, you can generate saliency maps to interpret the model's attention mechanism.

Figure 3.3 - Staircase detected output

Figure 3.4 - Captured Image

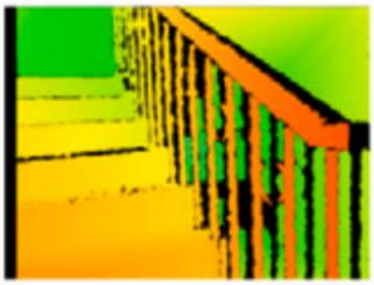


Figure 3.5 - Segmented Image

Figure 3.6 - Saliency Map Generation

The precision for the corridor class is the number of correctly predicted corridor photos (24) out of all predicted corridor photos (24+1=25), which amounts to 24/25=96%. So, 96% of the photos that our model classifies as corridor are actually corridor. On the other hand, the recall for corridor is the number of correctly predicted corridor photos (24) out of the number of actual corridor photos (24+2=26), which is 24/26=92.3%. This means that our classifier classified 92.3% of the corridor photos as corridor. The accuracy obtained for corridor is 96% and for staircase is 92%, and the overall accuracy for model is 94%. The accuracy can be improved in future with more training images of varying dimensions.

The results of the model on video input are shown. The model accurately identified corridors and staircases with a maximum speed of 14 images per second. The model speed can be improved by using more efficient processors. The accuracy obtained for corridor is 96% and for staircase is 92% and the overall accuracy for model is 94% with a maximum speed of 14 images per second.

CHAPTER 4

TARGET DETECTION

Convolutional neural networks (CNNs) have made significant advancements in several areas within the field of computer vision. Typically, contemporary detectors utilize pure convolution networks to draw out features. Traditional image classification networks like ResNet50 are used as the foundational architecture for our single object detectors. In the case of the YOLO series of detectors, they utilize a unique residual network known as Darknet, which offers superior efficiency in feature extraction. On the other hand, the Vision Transformer (ViT) represents a novel application of the Transformer model, which has become a favored choice for various natural language processing (NLP) tasks including machine translation, question answering, text classification, document summarization, and more. A crucial aspect of the Transformer's success lies in its capacity to comprehend intricate interdependencies among long input sequences through self-attention. With the introduction of the Vision Transformer (ViT), it has been demonstrated for the first time that the transformer architecture can be applied directly to image-based tasks. This is accomplished by perceiving an image as a series of patches and feeding these patches into an encoder network based on multi-headed self-attention layers. In our drone dataset, we only use a plain ViT-b16 model plus a few top layers to prove that it can achieve much higher accuracy than leading-edge convolutional networks VGG 16 and ResNet 50 at a cost of longer training time and larger model. By doing so, we demonstrate that while ViT can more efficiently capture long-range dependencies between image patches via self-attention, it also requires more training data and a careful design to perform well.

4.1 DATASET [2]

The drone dataset consists of 1359 images, 1297 of which contain only a single drone, while the remaining 62 images feature multiple drones. Along with these images, the dataset also provides the X and Y coordinates for the drone's center in each image, as well as the height and width necessary to draw bounding boxes for object detection. Currently, due to the limited number of multi-object training images and for the sake of simplicity, we are only using images containing a single drone for training.

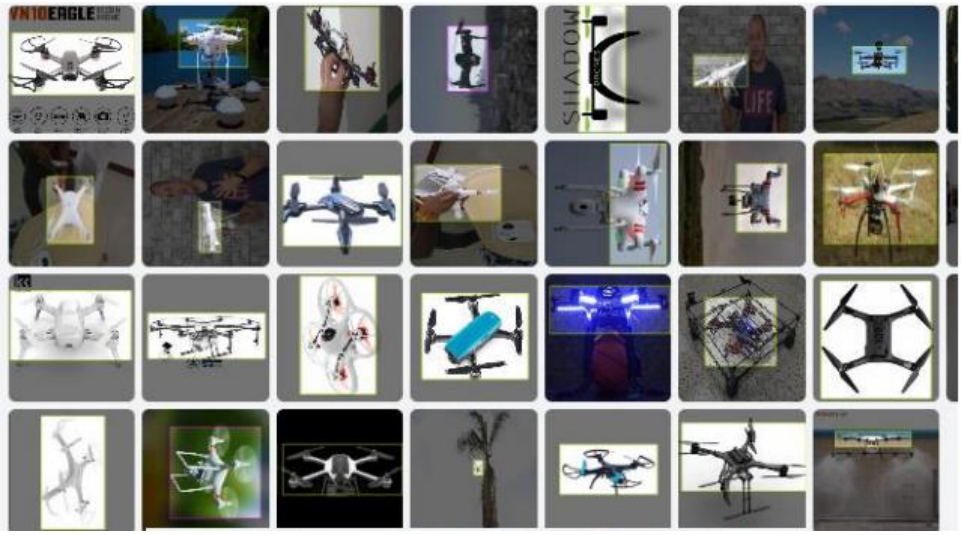


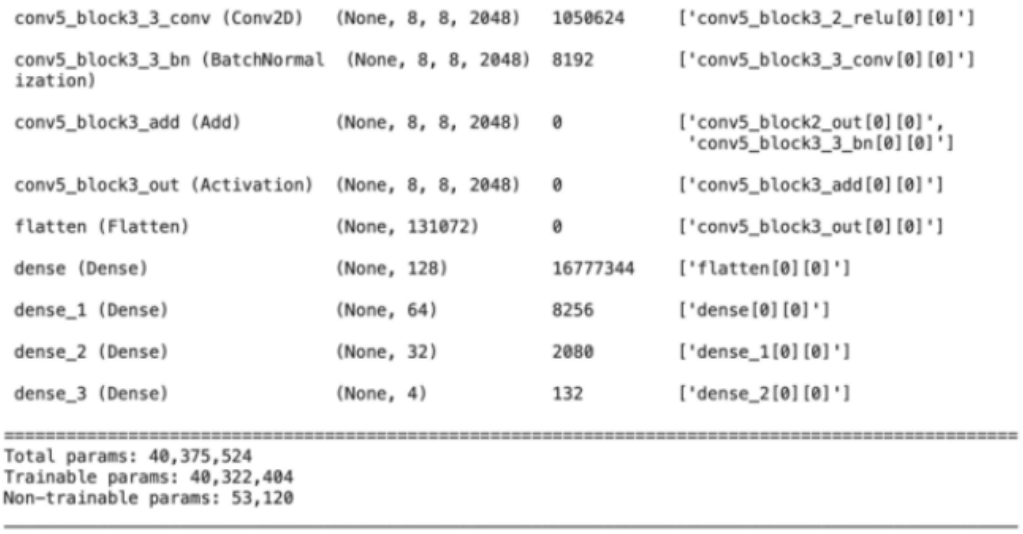
Figure 4.1 - Drone dataset

Analyzing the filtered dataset's complexity, we look at the X and Y coordinates of the center of the drone relative to the image for spatial complexity and the area ratio is defined as Equation 1 for perspective complexity. From the graphs, there appears to be a Gaussian distribution across spatial and perspective complexity in the drone dataset, showing that the filtered drone dataset images are not heavily skewed or biased by the Central Limit Theorem. The visualization also shows that a large portion of the drones in the dataset are very small and therefore hard to be located accurately. To test ViT's

full capability of multi-object detection, we further augment the dataset by applying mosaic, noise, flip, rotation and blur, so that the training dataset size increases to 3021 and there are more images containing multiple drones. We have trained YOLO v7, YOLO v8, and YOLOS on the augmented dataset.

4.2 SINGLE-DRONE DETECTION

A 14 layers vanilla customized CNN model and three popular transfer learning models are used as backbones to tackle the drone detection challenge: ResNet50, VGG16 and ViT-b16. We design the same top 5 layers for them for better comparison. Since it is a regression task, the MSE loss function and linear activation function are used on the final output layer. The Adam optimizer is deployed as usual with a learning rate of 0.0001. Notably, all the images are reshaped to (256,256,3) before training to fit in the vision transformer's unique patch size (16 by 16). Compared to classical CNN, ResNet and VGG network, the ViT-b16 transformer is more computationally intensive.



conv5_block3_3_conv (Conv2D)	(None, 8, 8, 2048)	1050624	['conv5_block3_2_relu[0][0]']
conv5_block3_3_bn (BatchNormalization)	(None, 8, 8, 2048)	8192	['conv5_block3_3_conv[0][0]']
conv5_block3_add (Add)	(None, 8, 8, 2048)	0	['conv5_block2_out[0][0]', 'conv5_block3_3_bn[0][0]']
conv5_block3_out (Activation)	(None, 8, 8, 2048)	0	['conv5_block3_add[0][0]']
flatten (Flatten)	(None, 131072)	0	['conv5_block3_out[0][0]']
dense (Dense)	(None, 128)	16777344	['flatten[0][0]']
dense_1 (Dense)	(None, 64)	8256	['dense[0][0]']
dense_2 (Dense)	(None, 32)	2080	['dense_1[0][0]']
dense_3 (Dense)	(None, 4)	132	['dense_2[0][0]']

Total params: 40,375,524
Trainable params: 40,322,404
Non-trainable params: 53,120

Figure 4.2 - ResNet50 model summary

Layer (type)	Output Shape	Param #
vit-b16 (Functional)	(None, 768)	85844736
flatten_3 (Flatten)	(None, 768)	0
dense_6 (Dense)	(None, 128)	98432
dense_7 (Dense)	(None, 64)	8256
dense_8 (Dense)	(None, 32)	2080
dense_9 (Dense)	(None, 4)	132

Total params: 85,953,636
Trainable params: 85,953,636
Non-trainable params: 0

Figure 4.3 - ViT-b16 model summary

Before analyzing the vision transformer, we need to first understand the working principles of the ViT-b16 model. An image is split into fixed-size patches, then linearly embedding and position embeddings are added to each of them. The necessity of the positional embeddings comes from the invariance of output context vector with respect to different permutation of input patches. In a self-attention layer, each output vector is calculated as a weighted sum of all value vectors, and the weights (i.e., attention scores) are based solely on the content of the key and query vectors and not their order. As a result, despite the output context vector order will be permuted in the same way as we permute the input sequence, the set content of output context vectors will remain the same. Then, the resulting sequence of vectors from embedding layers are fed to a standard Transformer encoder, which contains six transformer layers in our case. One transformer layer consists of one self-attention layer followed by a dense layer. Different from a CNN layer which only captures local context information by sliding a small kernel window, a self-attention layer treats all image patches equally, so it is inherently better at detecting long-range dependency between patches and requires a larger dataset

to efficiently learn these dependencies. To perform classification, we use the standard approach of adding an extra learnable "classification token" to the sequence. Unlike other transformers, the vision transformer does not have a decoder network, allowing our model to improve its training efficiency significantly, since we only need to train the embedding layers, a transformer encoder network, and a SoftMax classifier. In our case, we use 5 layers MLP with a bounding box detector head instead of a SoftMax classifier.

4.3 DRONE DETECTION

YOLO v7, YOLO v8 and YOLOS are trained on the augmented multi-drone dataset. Unlike the YOLO models which use CNNs as backbones and complex heads, YOLOS uses ViT as backbone and only an MLP of 2 hidden layers to implement both classification and bounding box regression heads. In essence, YOLOS is not optimized for better performance but to prove the transferability of ViT. The transformation from a Vision Transformer (ViT) to a YOLOS detector is straightforward. First, YOLOS eliminates the [CLS] token used for image classification and instead adds a hundred learnable detection tokens ([DET] tokens), which are randomly initialized, to the input patch embeddings ([PATCH] tokens) for the purpose of object detection. Secondly, during the training process, YOLOS substitutes the image classification loss found in ViT with a bipartite matching loss, enabling it to carry out object detection in a set prediction manner, in line with the method proposed by DETR, the first attempt to successfully apply transformers to multi-object detection.

4.4 DRONE PREDECTION OUTPUT

The Model is trained on training dataset of 3550 images for 300 epochs and validated using validation set of 1065 images.

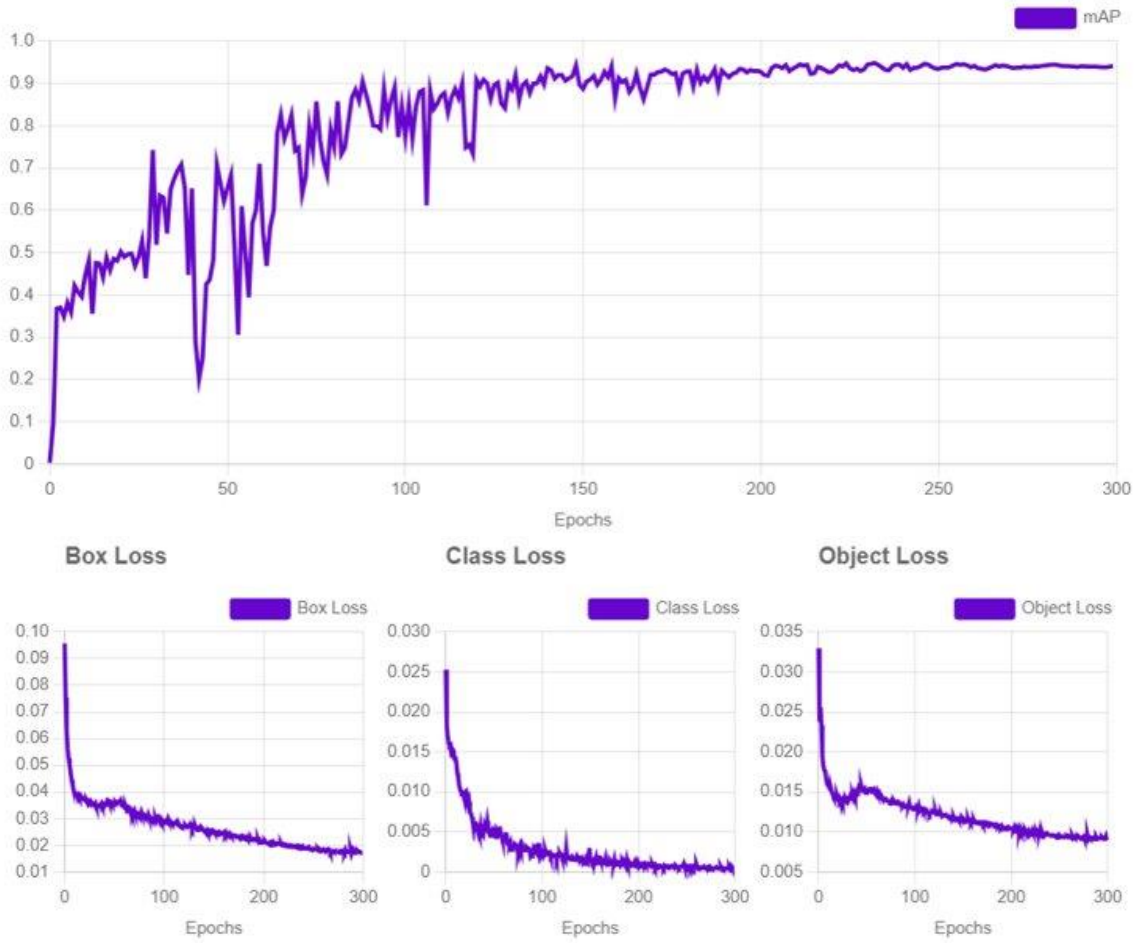


Figure 4.5 - YOLO prediction upto 300 epochs

In the initial epochs, the training loss typically decreases as the model learns to better predict bounding box coordinates and class labels. However, after a certain point, the model might start overfitting to the training data, leading to an increase in training loss. The validation loss helps assess the model's generalization to new data. Overfitting occurs when the model becomes too specialized to the training data and fails to generalize to new examples. The accuracy and loss function for both training and validation set is plotted for the number of epochs. The model converged after 130 epochs since the training accuracy had significantly increased after previous training.

Figure 4.6 - Real-time drone detection using NLP and CNN architecture

Project reveals another key advantage of vision transformers in object classification over CNNs. When the training dataset size is sufficiently large, ViT outperforms CNN by larger margins. This is intuitively understandable, given ViT's more complex model structure. It is also worthwhile to validate this in the object detection task, so we train our YOLOS model on the original dataset of only 1359 images without any data augmentation, with only 62 of them containing multiple drones in one image. The model converged after 130 epochs since the training accuracy had significantly increased after previous training. We found that the mAP drops quickly from 96% to around 92%, performing much closer to YOLO models (89%). From this, we conclude that the dataset size is extremely important for ViT-based models in object detection tasks.

CHAPTER 5
CONCLUSION AND FUTURE WORKS

5.1 CONCLUSION

In the comparative analysis conducted, we identified four main insights about the performance of Vision Transformer (ViT) based deep learning models in the realm of object detection. Primarily, ViT-based models consistently demonstrate superior performance over CNN-based models in similar epochs for object detection and classification tasks. This can be ascribed to the built-in self-attention mechanism in ViT that effectively captures long-range data dependencies. However, the use of ViT networks brings with it the demand for increased computational power and a lengthier training period, a stark contrast to the requirements of CNN networks.

With reference to indoor scene classification, the Model is trained on training dataset of 350 images for 50 epochs and validated using validation set of 120 images. The model accurately identified corridors and staircases with a maximum speed of 14 images per second. The model speed can be improved by using more efficient processors. The accuracy obtained for corridor is 96% and for staircase is 92% and the overall accuracy for model is 94% with a maximum speed of 14 images per second.

For target detection (drone) the model is trained on training dataset of 3550 images for 300 epochs and validated using validation set of 1065 images. We found that the mAP drops quickly from 96% to around 92%, performing much closer to YOLO models (89%).

This results in providing a competitive advantage for ViT over ResNet50 by 4.5 percent as VIT has 0 “non-trainable params” compared to ResNet50’s 53120 non-trainable params.

5.2 FUTURE WORKS

The idea is to leverage ViTs to analyze visual information from drone-mounted cameras, allowing the drone to understand its surroundings and make decisions for navigation. Optimizing the model for real-time inference on the drone's hardware will be executed by ensuring that the inference speed meets the requirements for responsive navigation. Future work will be on combining visual information from the ViT with data from other onboard sensors, such barometric sensors and IMU sensors with a developer board as a flight controller for a more comprehensive understanding of the environment.

REFERENCE

1. Ganesan, Anitha, and Anbarasu Balasubramanian. "Indoor versus outdoor scene recognition for navigation of a micro aerial vehicle using spatial color gist wavelet descriptors." Visual Computing for Industry, Biomedicine, and Art 2.1 (2019): 1-13.
2. Zhang, Junyang. "Towards a High-Performance Object Detector: Insights from Drone Detection Using ViT and CNN-based Deep Learning Models." 2023 IEEE International Conference on Sensors, Electronics and Computer Engineering (ICSECE). IEEE, 2023.
3. H. Jiang, J. Wang, Z. Yuan, Y. Wu, N. Zheng and S. Li, "Salient Object Detection: A Discriminative Regional Feature Integration Approach," 2013 IEEE Conference on Computer Vision and Pattern Recognition, Portland, OR, USA, 2013, pp. 2083-2090, doi: 10.1109/CVPR.2013.271.
4. Liu, Nian, et al. "Visual saliency transformer." Proceedings of the IEEE/CVF international conference on computer vision. 2021.
5. Ramachandran, Prajit, et al. "Stand-alone self-attention in vision models." Advances in neural information processing systems 32 (2019).
6. Bello, Irwan, et al. "Attention augmented convolutional networks." Proceedings of the IEEE/CVF international conference on computer vision. 2019.
7. Han, Kai, et al. "A survey on vision transformer." IEEE transactions on pattern analysis and machine intelligence 45.1 (2022): 87-110.
8. Kolesnikov, Alexander, et al. "Big transfer (bit): General visual representation learning." Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part V 16. Springer International Publishing, 2020.

9. Beyer, Lucas, et al. "Are we done with imagenet?." arXiv preprint arXiv:2006.07159 (2020).
10. Carion, Nicolas, et al. "End-to-end object detection with transformers." European conference on computer vision. Cham: Springer International Publishing, 2020.
11. Cordonnier, Jean-Baptiste, Andreas Loukas, and Martin Jaggi. "On the relationship between self-attention and convolutional layers." arXiv preprint arXiv:1911.03584 (2019).
12. Devlin, Jacob, et al. "Bert: Pre-training of deep bidirectional transformers for language understanding." arXiv preprint arXiv:1810.04805 (2018).
13. Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." Advances in neural information processing systems 25 (2012).
14. LeCun, Yann, et al. "Backpropagation applied to handwritten zip code recognition." Neural computation 1.4 (1989): 541-551.
15. Parkhi, Omkar M., et al. "Cats and dogs." 2012 IEEE conference on computer vision and pattern recognition. IEEE, 2012.
16. Zhai, Xiaohua, et al. "S4l: Self-supervised semi-supervised learning." Proceedings of the IEEE/CVF international conference on computer vision. 2019.
17. Zhao, Hengshuang, Jiaya Jia, and Vladlen Koltun. "Exploring self-attention for image recognition." Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. 2020.

APPENDIX I A: PREPARE DATASET

```
# Path to images and annotations
path_images = "..........."
path_annot = "..........."
path_to_downloaded_file = keras.utils.get_file(fname="caltech_101_zipped",
origin="...........",  extract=True, archive_format="zip", cache_dir="/")
download_base_dir = os.path.dirname(path_to_downloaded_file)
# Extracting tar files found inside main zip file
shutil.unpack_archive (os.path.join(download_base_dir, "...........", "..........."), ".")
shutil.unpack_archive (os.path.join(download_base_dir, "...........", "..........."), ".")
# list of paths to images and annotations
image_paths = [f for f in os.listdir(path_images) if
os.path.isfile(os.path.join(path_images, f))]
annot_paths = [f for f in os.listdir(path_annot) if os.path.isfile(os.path.join(path_annot,
f)) ]
image_paths.sort()
annot_paths.sort()
image_size = 224          # resize input images to this size
images, targets = [], []      # loop over the annotations and images, preprocess them and
store in lists
for i in range(0, len(annot_paths)):       # Access bounding box coordinates
   annot = scipy.io.loadmat(path_annot + annot_paths[i])["box_coord"][0]
   top_left_x, top_left_y = annot[2], annot[0]
   bottom_right_x, bottom_right_y = annot[3], annot[1]
   image = keras.utils.load_img (path_images + image_paths[i])
   (w, h) = image.size[:2]
   image = image.resize((image_size, image_size))       # resize images
   images.append(keras.utils.img_to_array(image))       # convert image to array and
append to list
# apply relative scaling to bounding boxes as per given image and append to list
targets.append((float(top_left_x)/w,float(top_left_y)/h,float(bottom_right_x)/w,float(b
ottom_right_y)/h))
# Convert the list to numpy array, split to train and test dataset
(x_train), (y_train)=(np.asarray(images[int(len(images)*0.8):]),
np.asarray(targets[int(len(targets)*0.8):]))
(x_test), (y_test)=(np.asarray(images[int(len(images)*0.8):]),
np.asarray(targets[int(len(targets)*0.8) :]),)
```

APPENDIX I B: IMPLEMENT THE PATCH CREATION LAYER

```
class Patches(layers.Layer):
    def __init__(self, patch_size):
        super().__init__()
        self.patch_size = patch_size

    def call(self, images):
        input_shape = ops.shape(images)
        batch_size = input_shape[0]
        height = input_shape[1]
        width = input_shape[2]
        channels = input_shape[3]
        num_patches_h = height // self.patch_size
        num_patches_w = width // self.patch_size
        patches = keras.ops.image.extract_patches(images, size=self.patch_size)
        patches = ops.reshape(
            patches,
            (
                batch_size,
                num_patches_h * num_patches_w,
                self.patch_size * self.patch_size * channels,
            )
        )
        return patches

    def get_config(self):
        config = super().get_config()
        config.update({"patch_size": self.patch_size})
        return config
```

APPENDIX I C: IMPLEMENT THE PATCH ENCODING LAYER

```
class PatchEncoder(layers.Layer):
    def __init__(self, num_patches, projection_dim):
        super().__init__()
        self.num_patches = num_patches
        self.projection = layers.Dense(units=projection_dim)
        self.position_embedding = layers.Embedding(
            input_dim=num_patches, output_dim=projection_dim
        )
    # Override function to avoid error while saving model
    def get_config(self):
        config = super().get_config().copy()
        config.update(
            {
                "input_shape": input_shape,
                "patch_size": patch_size,
                "num_patches": num_patches,
                "projection_dim": projection_dim,
                "num_heads": num_heads,
                "transformer_units": transformer_units,
                "transformer_layers": transformer_layers,
                "mlp_head_units": mlp_head_units,
            }
        )
        return config
    def call(self, patch):
        positions = ops.expand_dims(
            ops.arange(start=0, stop=self.num_patches, step=1), axis=0
        )
        projected_patches = self.projection(patch)
        encoded = projected_patches + self.position_embedding(positions)
        return encoded
```

APPENDIX I D: BUILD THE VIT MODEL

```
def create_vit_object_detector(
    input_shape,
    patch_size,
    num_patches,
    projection_dim,
    num_heads,
    transformer_units,
    transformer_layers,
    mlp_head_units,
):
    inputs = keras.Input(shape=input_shape)
    patches = Patches(patch_size)(inputs)       # Create patches
    #Encode patches
    encoded_patches=PatchEncoder(num_patches,projection_dim)(patches)
    # Create multiple layers of the Transformer block.
    for _ in range(transformer_layers):
        # Layer normalization 1.
        x1 = layers.LayerNormalization(epsilon=1e-6)(encoded_patches)
        # Create a multi-head attention layer.
        attention_output = layers.MultiHeadAttention(
            num_heads=num_heads, key_dim=projection_dim, dropout=0.1
        )(x1, x1)
        # Skip connection 1.
        x2 = layers.Add()([attention_output, encoded_patches])
        # Layer normalization 2.
        x3 = layers.LayerNormalization(epsilon=1e-6)(x2)
        # MLP
        x3 = mlp(x3, hidden_units=transformer_units, dropout_rate=0.1)
        # Skip connection 2.
        encoded_patches = layers.Add()([x3, x2])
    # Create a [batch_size, projection_dim] tensor.
    representation = layers.LayerNormalization(epsilon=1e-6)(encoded_patches)
    representation = layers.Flatten()(representation)
    representation = layers.Dropout(0.3)(representation)
    # Add MLP.
    features = mlp(representation, hidden_units=mlp_head_units, dropout_rate=0.3)
    bounding_box=layers.Dense(4)(features) # Final four neurons that output box
    return keras.Model(inputs=inputs, outputs=bounding_box)   # return Keras model.
```

APPENDIX I E: RUN THE EXPERIMENT

```
def run_experiment(model, learning_rate, weight_decay, batch_size, num_epochs):
optimizer=keras.optimizers.AdamW(learning_rate=learning_rate,weight_decay)
    # Compile model.
    model.compile(optimizer=optimizer, loss=keras.losses.MeanSquaredError())
    checkpoint_filepath = "vit_object_detector.weights.h5"
    checkpoint_callback = keras.callbacks.ModelCheckpoint(checkpoint_filepath,
       monitor="val_loss",save_best_only=True,save_weights_only=True,)
    history = model.fit
(x=x_train,y=y_train,batch_size=batch_size,epochs=num_epochs,validation_split=0.1,callbac
ks=[checkpoint_callback,keras.callbacks.EarlyStopping(monitor="val_loss", patience=10),],)
    return history
input_shape = (image_size, image_size, 3)   # input image shape
learning_rate = 0.001
weight_decay = 0.0001
batch_size = 32
num_epochs = 100
num_patches = (image_size // patch_size) ** 2
projection_dim = 64
num_heads = 4
transformer_units=[projection_dim*2,projection_dim,]#  Size of the transformer layers
transformer_layers = 4
mlp_head_units = [2048, 1024, 512, 64, 32]  # Size of the dense layers
history = []
num_patches = (image_size // patch_size) ** 2
vit_object_detector = create_vit_object_detector(input_shape,patch_size,
num_patches,projection_dim,num_heads,transformer_units,transformer_layers,
mlp_head_units,)
# Train model
history = run_experiment
(vit_object_detector, learning_rate, weight_decay, batch_size, num_epochs)
def plot_history(item):
    plt.plot(history.history[item], label=item)
    plt.plot(history.history["val_" + item], label="val_" + item)
    plt.xlabel("Epochs")
    plt.ylabel(item)
    plt.title("Train and Validation {} Over Epochs".format(item), fontsize=14)
    plt.legend()
    plt.grid()
    plt.show()
plot_history("loss")
```

APPENDIX I F: EVALUATE THE MODEL

```
import matplotlib.patches as patches
# Saves the model in current path
vit_object_detector.save("vit_object_detector.keras")
# To calculate IoU (intersection over union, given two bounding boxes)
def bounding_box_intersection_over_union(box_predicted, box_truth):
    # get (x, y) coordinates of intersection of bounding boxes
    top_x_intersect = max(box_predicted[0], box_truth[0])
    top_y_intersect = max(box_predicted[1], box_truth[1])
    bottom_x_intersect = min(box_predicted[2], box_truth[2])
    bottom_y_intersect = min(box_predicted[3], box_truth[3])
    # calculate area of the intersection bb (bounding box)
    intersection_area=max(0,bottom_x_intersect-top_x_intersect+1)*max(
        0, bottom_y_intersect - top_y_intersect + 1)
    # calculate area of the prediction bb and ground-truth bb
    box_predicted_area=(box_predicted[2]-box_predicted[0]+1)*(box_predicted[3]-
box_predicted[1]+1)
    box_truth_area = (box_truth[2]-box_truth[0]+1) * (box_truth[3] - box_truth[1] + 1)
# calculate intersection over union by taking intersection area and dividing it by the
sum of predicted bb and ground truth bb areas subtracted by the interesection area
return ioU
   return intersection_area/float(box_predicted_area+box_truth_area-intersection_area)
i, mean_iou = 0, 0
# Compare results for 10 images in the test set
for input_image in x_test[:10]:
    fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(15, 15))
    im = input_image
    # Display the image
    ax1.imshow(im.astype("uint8"))
    ax2.imshow(im.astype("uint8"))
    input_image = cv2.resize(
        input_image, (image_size, image_size), interpolation=cv2.INTER_AREA)
    input_image = np.expand_dims(input_image, axis=0)
    preds = vit_object_detector.predict(input_image)[0]
    (h, w) = (im).shape[0:2]
    top_left_x, top_left_y = int(preds[0] * w), int(preds[1] * h)
    bottom_right_x, bottom_right_y = int(preds[2] * w), int(preds[3] * h)
    box_predicted = [top_left_x, top_left_y, bottom_right_x, bottom_right_y]
    # Create the bounding box
    rect = patches.Rectangle(
```

```
        (top_left_x, top_left_y),bottom_right_x - top_left_x,bottom_right_y - top_left_y,
        facecolor="none",edgecolor="red",linewidth=1,)
    # Add the bounding box to the image
    ax1.add_patch(rect)
    ax1.set_xlabel(
        "Predicted: "
        + str(top_left_x)
        + ", "
        + str(top_left_y)
        + ", "
        + str(bottom_right_x)
        + ", "
        + str(bottom_right_y))
    top_left_x, top_left_y = int(y_test[i][0] * w), int(y_test[i][1] * h)
    bottom_right_x, bottom_right_y = int(y_test[i][2] * w), int(y_test[i][3] * h)
    box_truth = top_left_x, top_left_y, bottom_right_x, bottom_right_y
    mean_iou += bounding_box_intersection_over_union(box_predicted, box_truth)
    # Create the bounding box
    rect = patches.Rectangle(
        (top_left_x, top_left_y),bottom_right_x - top_left_x,bottom_right_y - top_left_y,
        facecolor="none",edgecolor="red",linewidth=1,)
    # Add the bounding box to the image
    ax2.add_patch(rect)
    ax2.set_xlabel(
        "Target: "
        + str(top_left_x)
        + ", "
        + str(top_left_y)
        + ", "
        + str(bottom_right_x)
        + ", "
        + str(bottom_right_y)
        + "\n"
        + "IoU"
        + str(bounding_box_intersection_over_union(box_predicted, box_truth)))
    i = i + 1
print("mean_iou: " + str(mean_iou / len(x_test[:10])))
plt.show()
```