Illinois Institute of Technology

CS 577 Deep Learning

Object Detection using Vision Transformers (ViT)

Sai Manohar Vemuri (A20514848)

Sesha Shai Datta Kolli (A20516330)

Aditya Shivakumar (A20513537)

Dr. Yan Yan

Table of Contents

Introduction

A crucial topic in Computer Vision, Object detection plays a key role in variety of sectors like autonomous vehicles, surveillance systems, health sector etc., Traditionally, Convolutional Neural Networks (CNNs) have been the go-to architecture for a lot of computer vision tasks including Object Detection. They were actually good in capturing local features, using convolutional layers to extract patterns and structures, which improves the accuracy and efficiency in these tasks. However, there is a significant breakthrough with the introduction of Vision Transformers, or ViTs. ViTs represent a promising alternative for various Object Detection task.

One of the game-changing features is their reliance on self-attention mechanisms. Unlike CNNs, which focus on local relationships, ViTs use mechanism self-attention to analyze complex relationships within an entire image. This approach provides a better understanding of the context, significantly enhancing their effectiveness in computer vision tasks.

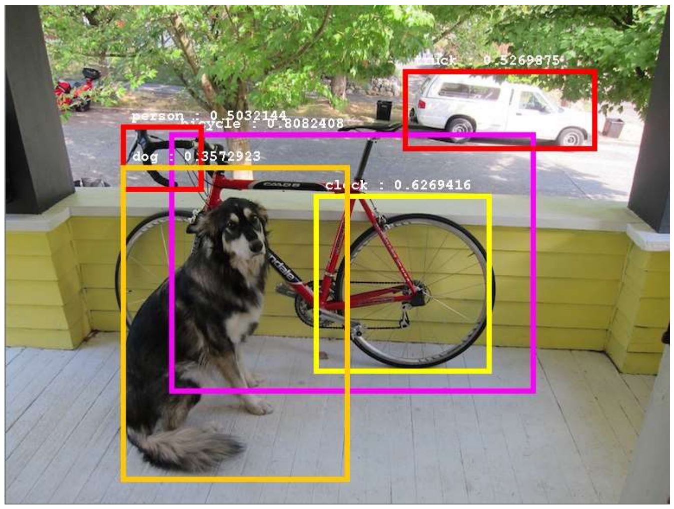


Fig 1.1. Object Detection Example

By dividing the images into fixed-size patches and encoding these patches into a series of embeddings., transformers efficiently capture spatial relationships, enabling effective feature extracting and used for tasks like object detection through mechanisms like self-attention and multi-head attention. This means that ViTs are not limited to examining local features; rather, they are able to understand the picture as an entire thing and identify complex relationships and long-range dependencies between different image parts.

Problem Description

This project suggests using Vision Transformers (ViTs) as a novel approach to address the challenge of creating an efficient and effective object detection system. The main goal is to make object detection and localization in images possible, with the intention of exceeding or at minimum equal the effectiveness of traditional Convolutional Neural Network (CNN)-based techniques.

The Vision Transformer is an innovative way of image processing that is different from convolutional layers. It treats each segment of an image as a separate token, conceiving an image as a collection of patches. These patches are encoded by applying transformer blocks later, which results in a classification output.

There are specific challenges when applying ViTs to more complex tasks like object detection. One major challenge is that high input resolution must be maintained. This requirement guarantees that the models accurately represent and capture the minute details present in images, which is essential for accurate detection. However, processing high-resolution images requires a significant amount of memory and processing power, which makes it difficult to implement ViTs efficiently.

The project's focus will be to optimize the ViT architecture for object detection in order to overcome these obstacles. This means optimizing the model's computational efficiency and adjusting it to accommodate a range of object sizes in images. In addition, approaches such as using more efficient self-attention mechanisms, including different training approaches will be taken into consideration.

To implement the ViT for object detection, the image is divided into fixed-size patches, each linearly embedded and supplemented with positional encodings. These embedding are then passed through transformer encoder block which consists of multihead self-attention layers which captures the contextual information and spatial relationship between these patches. This model is consisting of both classification head to predict the object class and regression head to predict the bounding box.

We have used Oxford IIT Pet dataset for implementing this project it contains around 4000 images. Each image contains the following annotations like species, breed, bounding boxes and trimaps. We decided to stick to the basics predicting cat or dog along with finding the bounding boxes due to limited computation resources.

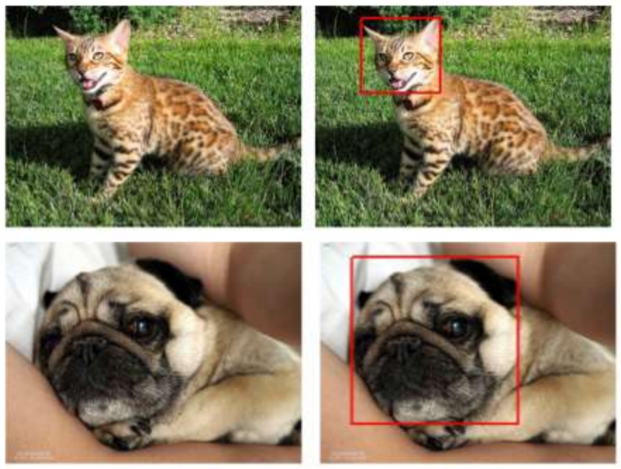


Fig 2.1. Dataset Sample

Literature Survey

"Beal et al. [1] implemented a model called ViT-FRCNN by modifying the original Vision Transformers model for the object detection task. They removed the final transformer state, which outputs the class, and replaced it with the layer that generates the spatial feature map. This feature map is then fed to the detection network, which consists of a Faster R-CNN model. It predicts the presence or absence of objects in the image using RPN. Later, the extracted features are ROI-pooled and then fed to a detection head, which regresses the bounding box coordinates and also predicts the class label."

Yanghao Li et al. [2] suggest that, since it is computationally expensive to train the model to compute global self-attention, it is advisable to use a pretrained model to perform global self-attention. They explored the use of restricted self-attention, where the feature map is divided into non-overlapping windows, and self-attention is computed within each window."

Liu et al. [6] addressed the data-hungry issue of existing detection transformers by modifying how key-value pairs are constructed in the cross-attention layer. They concluded that multi-scale deformable attention samples sparse features from local regions. They implemented sparse feature sampling using RoIAlign for local feature sampling and used layer-wise bounding box refinement to improve the detection performance. They also performed Label Augmentation to allow the model to receive more guidance from the data, potentially leading to faster and more effective training."

Methodology

We have used transformers model to implement this project and it involves mechanism called Self-attention which allows the model to extract contextual information. Key features of ViTs include the transformation of images into sequence of non-overlapping patches, and then these patches are encoded with positional embedding to preserve the spatial information. Then later these embeddings are passed to self-attention layer which captures the intrinsic relationship in the data. ViTs typically uses classification heads using a linear layer and a softmax activation function. But for obeject detection, both classification head and regression head is needed to predict object class and bounding box.

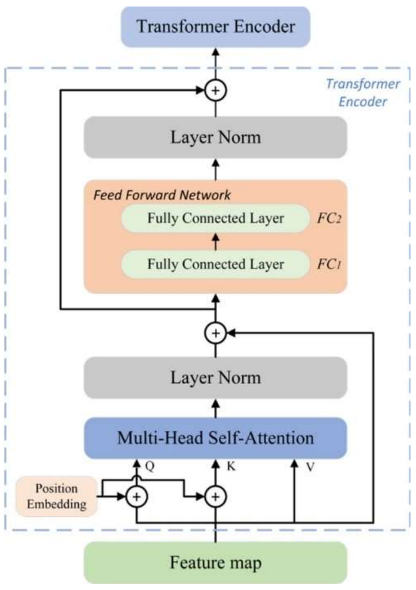


Fig 4.1. Architecture of Transformers Encoder block

Tokenization:

Images are divided into fixed size and non-overlapping patches representing visual token. This is done because self-attention mechanism in transformers works by considering the relationship between pairs of tokens in sequence. After tokenization self-attention can concentrate on connections between various local regions, allowing to capture both local and global patterns.

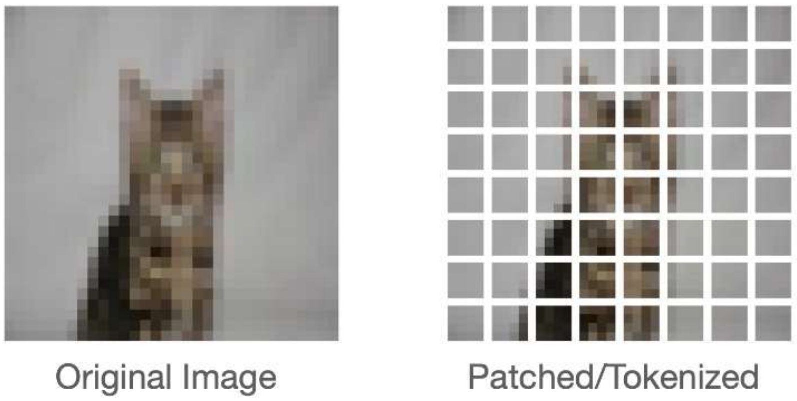


Fig 4.2. Tokenization Example

Embedding Layer:

The embedding layer in transformers is used to convert input patches into continuous vector space representation. Transformers use positional embeddings to understand the position of tokens in the sequence. The embedding layer in transformers is trainable and allows the model to adapt to its representation based on the task during the training process and data it encounters. These embeddings are then passed to self-attention block.

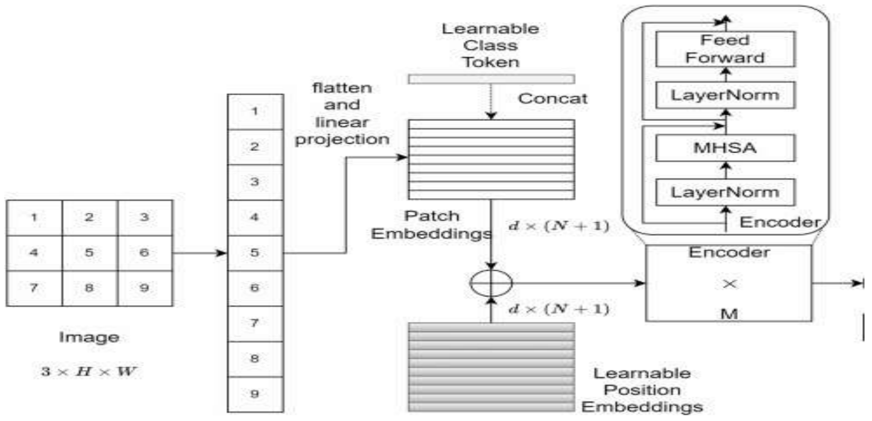


Fig 4.3. Positional Embedding Example

Self-attention:

A mechanism called self-attention, or intra-attention, enables each position in a sequence to pay attention to every other position in the same sequence. This allows the model to dynamically determine how important various parts of the input data points are. Each input embedding is transformed into Query (Q), which represents the element that is currently being focused on. Key (K), which represents the elements that are being compared against and Value (V), which represents the actual content of the elements. These vectors are obtained by multiplying the input embeddings by three weight matrices.

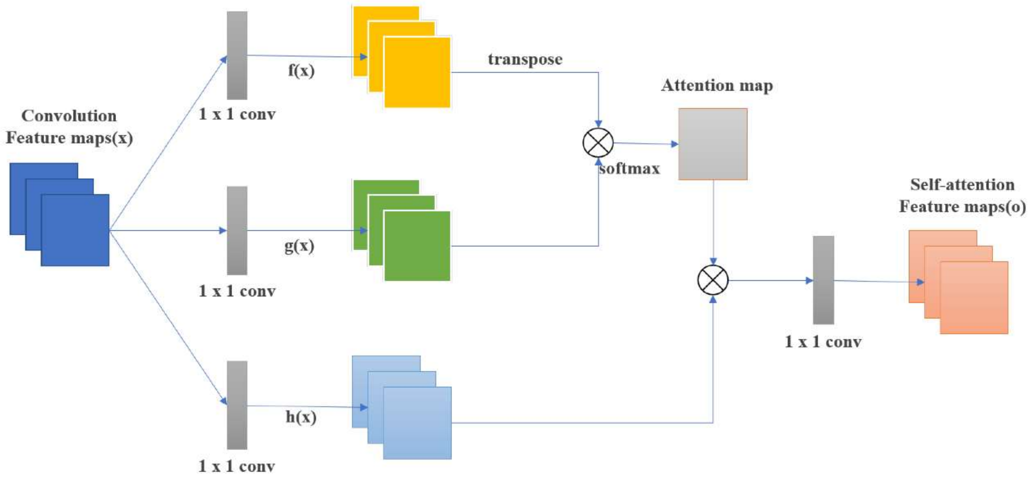


Fig 4.4. Self-Attention Layer Example

After that, self-attention calculates a score for every query and key combination to ascertain how other elements affect the element that is currently in focus. Typically, the dot product is used to calculate the score.

The square root of the key vectors' dimension is used to scale the scores down in order to stabilize the training process. To obtain a probability distribution, the scaled scores are subjected to a softmax function, which guarantees that the weights are positive and sum to one.

In transformers, self-attention helps the model look at the entire picture, focusing on how different parts relate to each other. It efficiently grasps the layout and relationships in the image, making it adaptable and flexible in understanding the overall context.

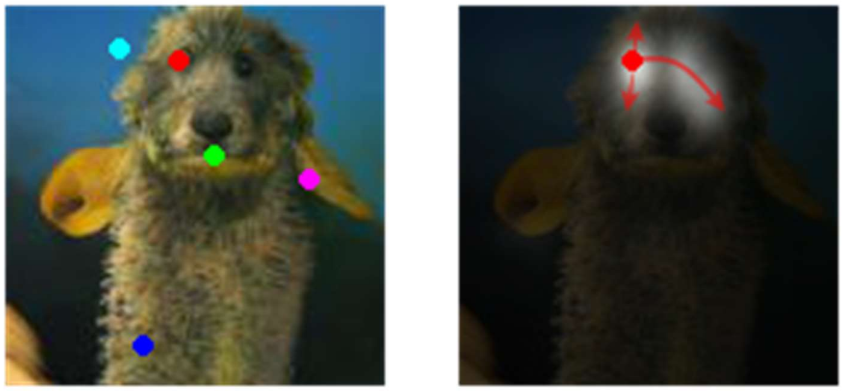


Fig 4.5. Self-Attention feature map example

MLP Block:

The Multi-Layer Perceptron (MLP) block within a transformer encoder plays a crucial role in processing individual token representations obtained from the self-attention mechanism. It encompasses a series of operations, including linear transformations, activation functions such as Gaussian Linear Units (GELU), and additional linear layers. Through this sequential process, non-linearity is introduced, enabling the model to discern intricate patterns within each token's representation and grasp complex relationships in the data. The MLP block adopts a position-wise feedforward approach, independently handling each token to capture local patterns and dependencies. Integration of a residual connection safeguards the original token representations, addressing potential vanishing gradient concerns. The resulting output undergoes normalization, promoting stable training. In essence, the MLP block enhances the transformer's ability to capture token-level features, refine representations, and excel in tasks spanning natural language processing and computer vision.

Implementation

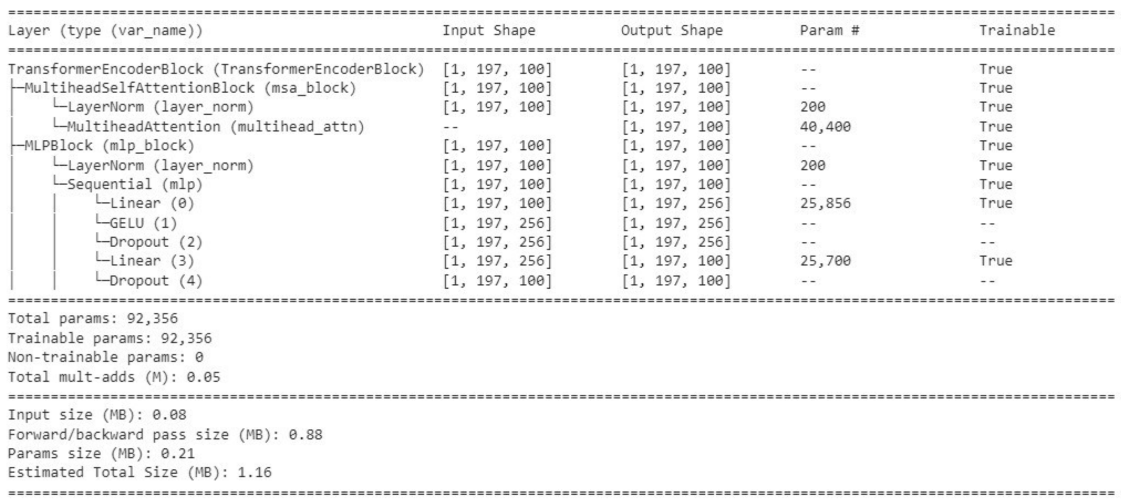
Data Preprocessing:

For preparing data for an object detection vision transformers model, I followed a series of steps. First, I resized the images and adjusted the bounding boxes accordingly to ensure uniformity in the dataset and also taking care of the computational requirements. Next, I normalized both the resized images and their corresponding bounding boxes. This normalization step is crucial for standardizing the input data, ensuring that the model can effectively learn from features within a consistent range. Finally, I organized the preprocessed data and saved it in a pickle file, making it convenient for later use in Google Colab. Instead of uploading the dataset in the google drive which is computationally very expensive for I/O operations, we have decided to do all the necessary preprocessing in the local machine and then uploaded a single pickle format file to google drive. This streamlined data preprocessing helps set the stage for training a vision transformer model on object detection tasks, enhancing model performance and facilitating seamless integration into the Google Colab environment.

Model Architecture:

In the Transformer model's encoder block, a single layer is dedicated to refining the representation of input data. This layer takes in a batch consisting of 197 tokens, with each token encoded into a 100-dimensional vector. The layer's architecture is anchored by two principal components. The first, a Multi-head Self-Attention Block, allows for the interaction of each token with every other, facilitating a deepened understanding of their interrelationships. Maintaining the input's dimensions, it optimizes 40,400 parameters through training to sharpen the model's performance. Following the self-attention process, the data progresses to an MLP Block, a multi-layer perceptron with layers that feature nonlinear activation functions and dropout for regularization. This segment of the layer expands the feature dimensions to 256 and then contracts them back to 100, employing 25,856 and 25,700 parameters for its two layers, respectively. Preceding each significant component, Layer Normalization is applied to standardize the data, thus aiding in stabilizing the learning trajectory. Altogether, the layer comprises 92,356 trainable parameters, all of which are adjusted during training to minimize prediction errors and enhance the accuracy of the model.

The following architecture represents single transformer layer:



```
========================================================================================================================
Layer (type (var_name))                          Input Shape          Output Shape         Param #              Trainable
========================================================================================================================
TransformerEncoderBlock (TransformerEncoderBlock) [1, 197, 100]       [1, 197, 100]        --                   True
├─MultiheadSelfAttentionBlock (msa_block)        [1, 197, 100]        [1, 197, 100]        --                   True
│    └─LayerNorm (layer_norm)                    [1, 197, 100]        [1, 197, 100]        200                  True
│    └─MultiheadAttention (multihead_attn)       --                   [1, 197, 100]        40,400               True
├─MLPBlock (mlp_block)                           [1, 197, 100]        [1, 197, 100]        --                   True
│    └─LayerNorm (layer_norm)                    [1, 197, 100]        [1, 197, 100]        200                  True
│    └─Sequential (mlp)                          [1, 197, 100]        [1, 197, 100]        --                   True
│    │    └─Linear (0)                           [1, 197, 100]        [1, 197, 256]        25,856               True
│    │    └─GELU (1)                             [1, 197, 256]        [1, 197, 256]        --                   --
│    │    └─Dropout (2)                          [1, 197, 256]        [1, 197, 256]        --                   --
│    │    └─Linear (3)                           [1, 197, 256]        [1, 197, 100]        25,700               True
│    │    └─Dropout (4)                          [1, 197, 100]        [1, 197, 100]        --                   --
========================================================================================================================
Total params: 92,356
Trainable params: 92,356
Non-trainable params: 0
Total mult-adds (M): 0.05
========================================================================================================================
Input size (MB): 0.08
Forward/backward pass size (MB): 0.88
Params size (MB): 0.21
Estimated Total Size (MB): 1.16
========================================================================================================================
```

We have used 4 transformer layers for this project with each consisting of 8 multi-head self-attention blocks.

1. Model Architecture Configuration: I've configured a vision transformer (ViT) with specific parameters to suit my task. The training resolution is set at 224x224 pixels in accordance with recommendations from the ViT paper. Since I'm dealing with RGB color images, the model expects input with 3 channels.

2. Image Patching Strategy: To facilitate the application of the transformer architecture, I've chosen a patch size of 8x8 pixels. This approach involves dividing images into patches, enabling the model to process them in a sequence-like manner.

3. Transformer Layers and Embedding: For the transformer layers, I've opted for 4, each incorporating self-attention mechanisms to capture relationships between different patches and positions within the image. The hidden size or embedding dimension is set to 128, influencing the model's capacity to represent features.

4. MLP Layers and Size: The Multi-Layer Perceptron (MLP) layers, crucial for processing token embeddings, have a size of 3096. These layers contribute to the model's ability to learn complex patterns within the data.

5. Self-Attention Heads: To enhance the model's capacity to capture diverse features and relationships, I've included 8 self-attention heads in each transformer layer. This allows the model to attend to different aspects of the input simultaneously.

6. Dropout for Regularization: To prevent overfitting during training, dropout is applied at various stages. Attention dropout, MLP dropout, and embedding dropout are set to 0.2, indicating a 20% dropout probability.

7. Classifier Heads: The model has separate heads for bounding box regression and classification. The heads consist of linear layers and normalization, and they output predictions for bounding box coordinates and class probabilities.

8. No. of classes: The model is designed for a classification task with 2 classes, and this configuration aligns with the ViT paper's architecture. These parameters can be further adjusted based on the characteristics of the dataset and the specific requirements of my task.

9. Sigmoid Activation: The class predictions are passed through a sigmoid activation function to ensure that the output is in the range [0, 1], suitable for representing probabilities.

Hyperparameters:

Number of Epochs:

This represents the number of times the entire dataset is passed through the neural network during training. In this case, the training process will iterate over the entire dataset 75 times.

Optimizer:

Initially, the Adam optimizer was employed for updating the model parameters during training. However, after experimentation, it was observed that using the AdamW optimizer yielded better results. AdamW is an extension of Adam that incorporates weight decay, which penalizes large weights. This modification helps in controlling overfitting and improving the generalization performance of the model. Therefore, the AdamW optimizer with a learning rate of 0.001 and weight decay of 0.0001 is chosen for the training process.

Learning Rate Scheduler:

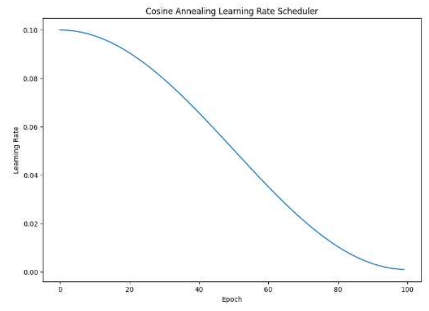


Fig 5.1. Cosine Annealing Example

The learning rate scheduler adjusts the learning rate during training. The Cosine Annealing LR scheduler reduces the learning rate in a cosine annealing manner. T_max=75 indicates the number of epochs for a full cosine annealing cycle, and eta_min=0.00001 is the minimum learning rate.

$$\eta_t = \eta_{min}^i + \frac{1}{2}\left(\eta_{max}^i - \eta_{min}^i\right)\left(1 + \cos\left(\frac{T_{cur}}{T_i}\pi\right)\right)$$

In the above formula, "n_min" and "n_max" represent the ranges for the learning rate, taking into consideration the number of epochs completed since the last restart.

Batch Size:

Batch size represents the number of training examples utilized in one iteration. In this case, the model is trained on batches of 16 samples at a time.

Loss Functions:

The model is trained using a combination of Binary Cross Entropy (BCE) and Mean Squared Error (MSE) loss functions, representing a multi-task learning approach. This allows the model to simultaneously optimize for binary classification tasks, such as object class (BCE), and regression tasks, such as bounding box values (MSE). The combination of these loss functions enhances the model's capacity to handle both classification and detection aspects, making it well-suited for object detection tasks where identifying objects and precisely locating them within an image are crucial.

Loss = BCE + MSE

The Binary Cross Entropy Loss (BCE) is commonly used for binary classification tasks, and it measures the difference between predicted and target class probabilities.

$$L_{BCE} = -\frac{1}{n}\sum_{i=1}^{n}(Y_i \cdot \log \hat{Y}_i + (1 - Y_i) \cdot \log(1 - \hat{Y}_i))$$

The Mean Squared Error Loss (MSE) is often used for regression tasks, measuring the average squared differences between predicted and target values.

$$\text{MSE} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

These hyperparameters collectively define the training setup, specifying how the model is optimized, how the learning rate changes over epochs, the batch size for training, and the loss functions used to calculate the error during training.

Evaluation Metrics

Accuracy Plot: (Classification Accuracy)

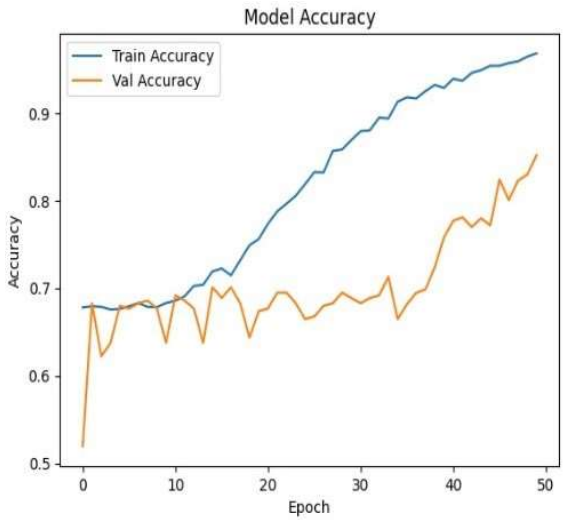


Fig 6.1. Model Accuracy

Training Accuracy (Blue Line):
The ratio of accurately predicted bounding boxes to the total number of training samples is displayed by the training accuracy (blue line). The steady rise suggests that throughout training data epochs, the model is getting increasingly accurate.

Validation accuracy (Orange Line): Shows the correctness of the model using a different validation dataset. The validation accuracy plateau towards later epochs indicates that the model may be reaching its maximum generalization potential and that additional training may converge.

mIOU Plot:

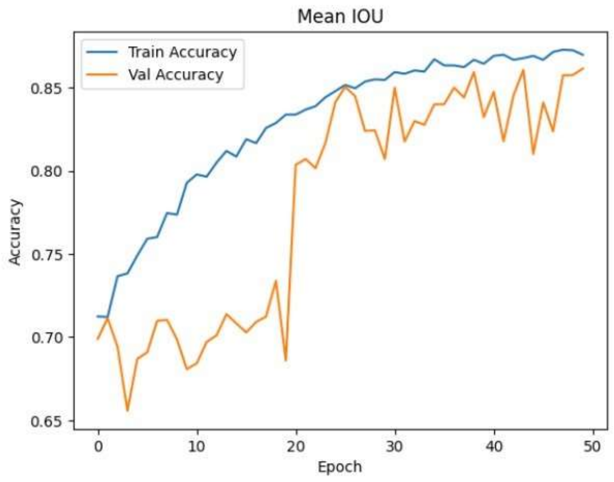


Fig 6.2. Mean IOU

Training mIOU (Blue Line): represents the mean mIOU computed from the training dataset's various classes. A rising trend suggests that the segmentation performance of the model is getting better.

Validation mIOU (Orange Line): shows the mean mIOU for a specific validation set. The validation mIOU plateau suggests that the model is reaching saturation and that more training epochs could produce appreciable gains.

Loss Graph:

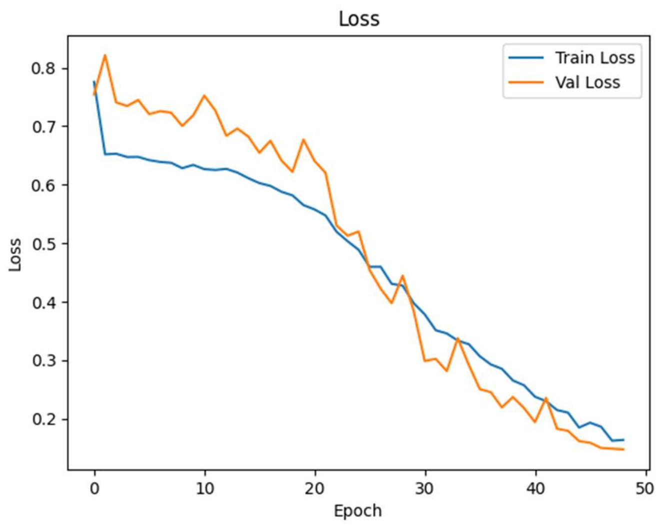


Fig 6.3. Model Loss

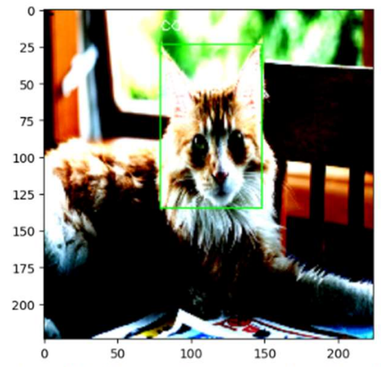
Training Loss (Blue Line): Shows the average loss calculated for each batch in each epoch during the training phase. The trend toward decrease suggests that the model is successfully reducing the discrepancy between the ground truth annotations and anticipated bounding boxes.

The validation loss, represented by the orange line, is the mean loss on an independent validation dataset. Variations in the validation loss indicate problems with the model's ability to generalize to new data. An indication of overfitting or a point at which more training yields declining returns could be indicated by the initial decline followed by stabilization.

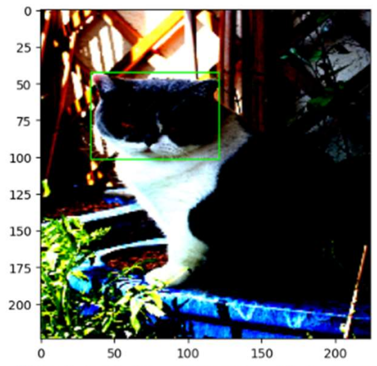
Results

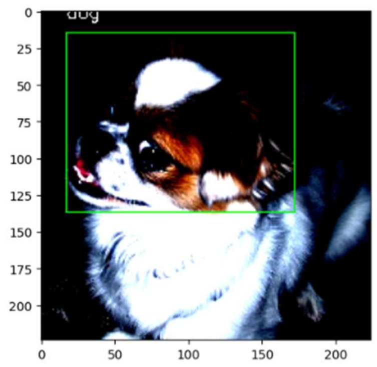
ACTUAL **ACTUAL** **ACTUAL**

```
WARNING:matplotlib.image:Clipping input data to the valid
tensor([ 79.,  24., 148., 135.])
```

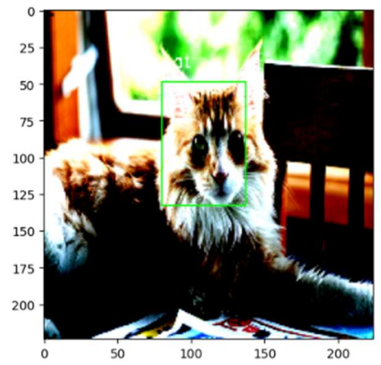


```
WARNING:matplotlib.image:Clipping input data to the valid
tensor([ 34.0000,  43.0000, 121.0000, 102.0000])
```

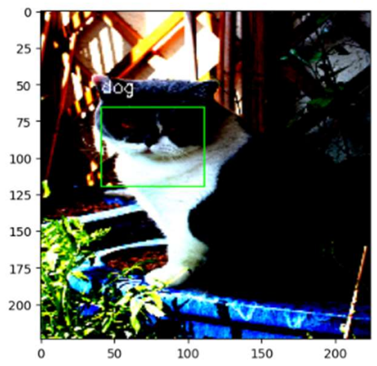




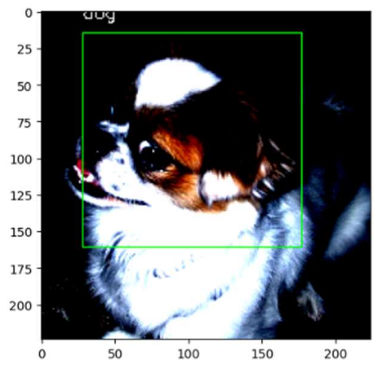
```
WARNING:matplotlib.image:Clipping input data to the valid
tensor([ 80.5374,  49.6888, 137.4629, 133.2605], device='c
```



```
WARNING:matplotlib.image:Clipping input data to the valid
tensor([ 41.3562,  66.6722, 111.6269, 120.8613], device='c
```



```
ARNING:matplotlib.image:Clipping input data to the valid ra
ensor([ 28.0703,  15.3998, 177.0660, 161.6669], device='cud
```

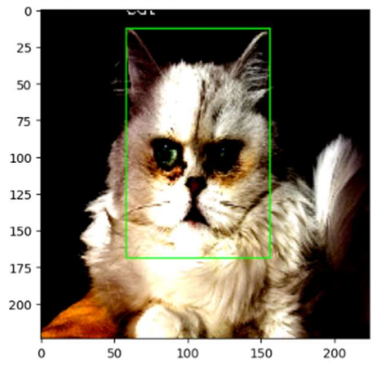


PREDICTED **PREDICTED** **PREDICTED**

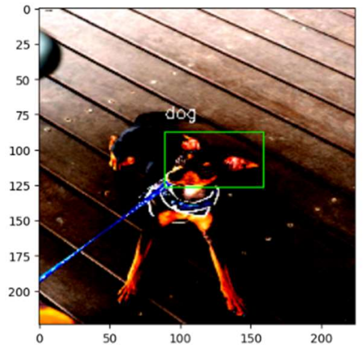
(correct class prediction) **(wrong class prediction)** **(correct class prediction)**

ACTUAL **ACTUAL**

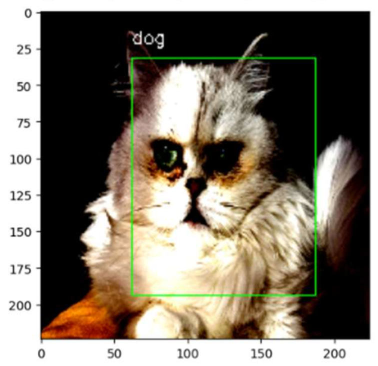
```
WARNING:matplotlib.image:Clipping input data to the valid r
tensor([ 58.,  13., 156., 169.])
```



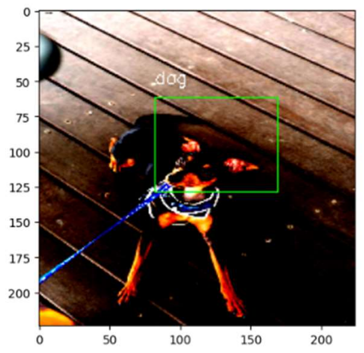
```
WARNING:matplotlib.image:Clipping input data to the valid ra
tensor([ 89.,  88., 159., 127.])
```



```
WARNING:matplotlib.image:Clipping input data to the valid r
tensor([ 62.7384,  32.4548, 187.6617, 194.4282], device='cu
```



```
WARNING:matplotlib.image:Clipping input data to the valid ra
tensor([ 82.7489,  62.9836, 169.1888, 129.3767], device='cuc
```

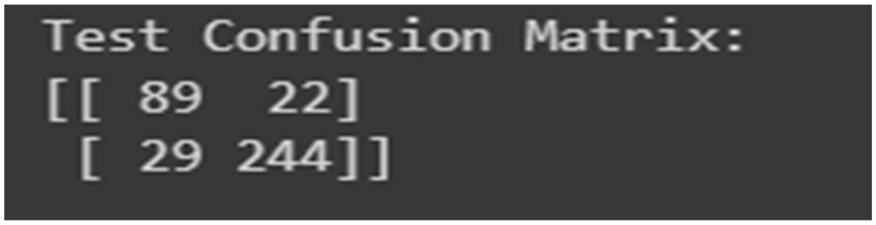


PREDICTED **PREDICTED**

(wrong class prediction) **(correct class prediction)**

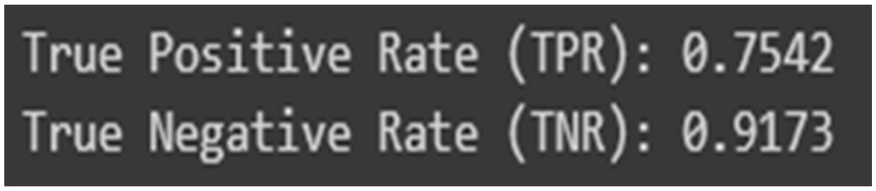
Classification Metrics

Confusion Matrix:



```
Test Confusion Matrix:
[[ 89  22]
 [ 29 244]]
```

TPR & FPR:



```
True Positive Rate (TPR): 0.7542
True Negative Rate (TNR): 0.9173
```

This indicates that our model is predicting one class more than the other one. Since there is a class imbalance problem in our dataset, this seems usual and we can overcome this by using weighted loss functions like Focal Loss or Weighted binary cross entropy loss which gives more weightage to the minority class. Or we can even oversample or duplicate the instances of minority class. But training it further may lead to overfitting. So, in future work we might solve this issue with the class imbalance problem.

Link to the project Repo:

https://github.com/mvemuri6642/Object_Detection_using_Transformers

Conclusion:

In conclusion, we have built a simple transformers model and tuned it for object detection task along with predicting the object class. We choose Oxford IIIT Pet dataset to train this model and stick to the basics of predicting only either cat or dog since we don't have enough computational resources. We have tried different model parameters and finally was able to build an efficient model with limited training resources. We have built a custom architecture from scratch designed for this specific task. Due to limited computational resources, we have carefully finetuned model on our dataset relevant to the task. We have experimented with different optimizers, learning rates, etc., we have evaluated the model using metrics like accuracy for classification problem and mean IOU for the bounding box prediction problem. This iterative process allowed us to refine the model and choose the optimal set of parameters that resulted in better performing model even though we had limited resources.

Contributions:

Name	Contribution
Sai Manohar Vemuri	Modelling, Data Preprocessing
Sesha Shai Datta Kolli	Modelling, Evaluation
Aditya Shivakumar	Modelling, Documentation, Literature survey

***All three of us worked on model architecture design with different parameters and tested in different environments.**

Future work:

In future work, it would be useful to explore the model scalability and generalization capabilities by adjusting hyperparameters. Furthermore, evaluating the model's performance on larger dataset like COCO might improve the model's capacity to handle the variety of scenarios like developing the model to recognize more object classes and finding more than 1 object in an image. Solving the class imbalance problem, we faced in this dataset by using different strategies. Additionally, by extending the model capability to address a multi-class object detection task will be potentially useful and holds the promise of expanding its practical applicability.

References:

[1]. Beal, Josh, Eric Kim, Eric Tzeng, Dong Huk Park, Andrew Zhai and Dmitry Kislyuk. “Toward Transformer-Based Object Detection.” ArXiv abs/2012.09958 (2020)

[2]. Li, Yanghao & Mao, Hanzi & Girshick, Ross & He, Kaiming. (2022). Exploring Plain Vision Transformer Backbones for Object Detection. 10.1007/978-3-031-20077-9_17.

[3]. Dosovitskiy, Alexey, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani et al. "An image is worth 16x16 words: Transformers for image recognition at scale." arXiv preprint arXiv:2010.11929 (2020).

[4]. Vaswani, Ashish, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. "Attention is all you need." Advances in neural information processing systems 30 (2017).

[5]. Carion, Nicolas, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. "End-to-end object detection with transformers." In European conference on computer vision, pp. 213-229. Cham: Springer International Publishing, 2020.

[6]. Liu, Yahui, Enver Sangineto, Wei Bi, Nicu Sebe, Bruno Lepri, and Marco Nadai. "Efficient training of visual transformers with small datasets." Advances in Neural Information Processing Systems 34 (2021): 23818-23830.