Comparative study of Object Detection performances between CNN and Transformer based models

Nishanth Rachakonda
Department of Computer Science
University of California, San Diego
nrachakonda@ucsd.edu

Soumya Ganguly
Department of Mathematics
University of California, San Diego
s1gangul@ucsd.edu

Krishna Daamini Ellendula
Department of Electrical & Computer Engineering
University of California, San Diego
kellendula@ucsd.edu

Pranav Khanna
Department of Computer Science
University of California, San Diego
pkhanna@ucsd.edu

Abstract

This project paper provides a comprehensive analysis of deep learning techniques for object detection, with a particular emphasis on comparing the efficiency of CNN-based models and attention-based Transformer models. The study investigates various model architectures, like Resnet, VGG, and Visual Transformer, all of which utilize the Faster RCNN framework. The experimental evaluations are conducted on the widely used Pascal VOC dataset where some of the models were pre-trained on the COCO dataset. With untrained backbones, the best results obtained were from a 'Faster R-CNN with ViT backbone and corners aligned', with 2.1 mAP, and with pre-trained backbones (on COCO dataset) the best results obtained were from 'Faster R-CNN with pre-trained ResNet backbone', with 24 mAP.

1 Introduction:

Object detection is a fundamental problem in computer vision that involves localizing and classifying objects of interest within an image. It has numerous real-world applications, such as surveillance systems, autonomous vehicles, and medical imaging. The goal of object detection is to identify the presence and location of objects within an image and assign them to predefined categories. The problem can be further subdivided into two main tasks: object localization and object classification. Object localization refers to the process of determining the precise location of objects within an image, usually by drawing a bounding box around them. This is necessary for distinguishing multiple objects in an image and understanding their spatial relationship. Object classification, on the other hand, involves assigning a label or category to each object detected in an image. This task is often performed after the objects have been localized, but can also be done simultaneously with localization.

Object detection is a challenging problem because objects in real-world images can vary significantly in appearance, scale, and orientation. Additionally, objects may be occluded or partially obscured by other objects or background clutter. Therefore, a successful object detection algorithm must be able to handle such variations while maintaining high accuracy and computational efficiency.

CNN-based networks have been among the earliest approaches used in deep learning-based object detection. These networks generate feature maps from input images, which are subsequently fed to object detection algorithms to identify objects. In this study, we aimed to compare the performance

of attention-based networks with convolution-based feature encoding and object detection heads. To this end, we utilized the Vision Transformer (ViT) model to generate feature encodings for images and evaluated its performance against ResNet and VGG models. Furthermore, we compared the performance of attention-based object detectors with that of CNN-based models.

We conducted experiments using Faster R-CNN models with CNN and ViT backbones along with end-to-end ViT models. With untrained backbones, the best results obtained were from a 'Faster R-CNN with ViT backbone and corners aligned', with 2.1 mAP, and with pre-trained backbones (on COCO dataset) the best results obtained were from 'Faster R-CNN with pre-trained ResNet backbone', with 24 mAP.

2 Related Work:

Object detection is an active research topic in computer vision, and a wide range of deep-learning models have been proposed to tackle this problem. Ren et al. [9] introduced the Faster R-CNN model, which achieved state-of-the-art results on several benchmark datasets. Redmon et al. [7] proposed the You Only Look Once (YOLO) model, which can achieve real-time object detection with high accuracy. Liu et al. [6] introduced the Single Shot Multibox Detector (SSD), which is another popular model for object detection.

The effectiveness of deep residual networks for image recognition has been demonstrated by He et al. [5]. Szegedy et al. [11] proposed the GoogLeNet architecture, and Simonyan and Zisserman [10] introduced the VGG network, which have achieved state-of-the-art results on several image recognition benchmarks. Recently, attention-based Transformer models have shown promising results in various computer vision tasks. Carion et al. [1] proposed an end-to-end object detection model based on the Transformer architecture, which achieved state-of-the-art performance on the COCO dataset. Zheng et al. [13] explored the use of Transformer models for semantic segmentation, and Chen et al. [2] proposed the Visual Transformer model for object detection, which achieved state-of-the-art performance on the COCO dataset. In this work, we focus on comparing the efficiency of CNN-based models and attention-based Transformer models for object detection, with a specific focus on the Faster RCNN framework and different model architectures like Resnet, VGG, and Visual Transformer.

3 Background Material:

3.1 Datasets:

In this project, till now we have worked with both PASCAL VOC and COCO datasets. The PASCAL VOC-2007 dataset is a huge dataset with images of everyday objects. It is a dataset that is used for image classification and object detection. We use this dataset to apply object detection so that we can recognize objects from a number of visual object classes in realistic scenes. The twenty object classes present in this dataset are:

- *Human:* Person
- *Animal:* Bird, Cat, Cow, Dog, Horse, Sheep
- *Vehicle:* Aeroplane, Bicycle, Boat, Bus, Car, Motorbike, Train
- *Indoor:* Bottle, Chair, Dining table, Potted plant, Sofa, Tv/Monitor

We are also considering the class 'background' as a separate class from the object classes above which makes a total of 21 object classes in this dataset. The data has been split into 50% for training/validation and 50% for testing. The distributions of images and objects by class are approximately equal across the training/validation and test sets. In total there are 9,963 images, containing 24,640 annotated objects.

On the other hand, the COCO dataset (short for Common Objects in Context) is a widely used benchmark dataset for object detection, segmentation, and captioning tasks. It was created by Microsoft in collaboration with several academic institutions and is maintained by the COCO Consortium. The COCO dataset contains around 164,000 annotated images, annotated with object bounding boxes, segmentation masks, and captions. These images are split into a training set of 118,000 images,

which are used to train object detection and segmentation models, a validation set of 5,000 images, which are used to tune the hyperparameters of models during training, evaluate their performance, etc. and a test set of 20,000 images, which are used to evaluate the performance of trained models in a blind test.

The COCO dataset includes annotations for 80 object classes, such as a person, car, bicycle, dog, etc. Here also we add a 'background' class to make the total number of object classes to be 81. These classes were selected to represent a broad range of objects that are commonly encountered in daily life. Each object instance in the dataset is annotated with a label indicating its object class, as well as a bounding box or segmentation mask that outlines the extent of the object in the image.

The COCO dataset has become a popular benchmark for evaluating the performance of object detection and segmentation models and has been used in numerous research papers and competitions.

3.2 Faster R-CNN :

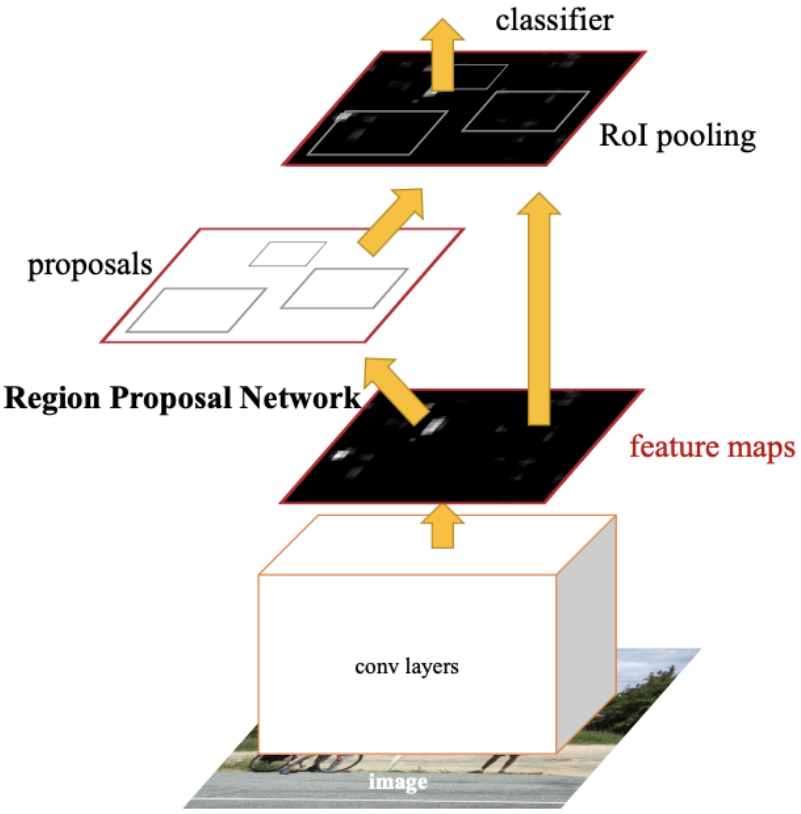

Figure 1: Faster R-CNN Architecture (image courtesy: [9])

Faster R-CNN (Region-based Convolutional Neural Network) is a popular object detection algorithm that uses a two-stage approach to detect objects in images.

The architecture of Faster R-CNN consists of three main components: a convolutional neural network (CNN) backbone, a Region Proposal Network (RPN), and a Region of Interest (RoI) head. Here's a brief overview of each component:

- **CNN Backbone :** The CNN backbone is responsible for extracting features from the input image. Faster R-CNN typically uses a pre-trained CNN, such as VGG or ResNet, which

has been trained on a large dataset, such as ImageNet, to extract high-level features from images.

- **Region Proposal Network (RPN) :** The RPN is a neural network that operates on the CNN feature map and generates object proposals, which are regions in the image that are likely to contain an object. The RPN accomplishes this by sliding a small network, called an anchor, over the feature map and predicting objectness scores and bounding box offsets for each anchor. The objectness score indicates the probability that an object is present in the anchor, and the bounding box offsets specify how the anchor should be adjusted to tightly fit the object.
- **Region of Interest (RoI) Head :** The RoI head takes the proposed regions generated by the RPN and extracts features from them using the CNN backbone. These features are then fed into a classifier and a regressor, which predict the class label and refine the bounding box coordinates, respectively.

During training, Faster R-CNN is trained end-to-end using a multi-task loss function that combines the losses from the RPN, classifier, and regressor. The RPN loss encourages the network to propose accurate object regions, while the classifier and regressor losses encourage the network to correctly classify objects and refine their bounding boxes.

Overall, Faster R-CNN is a powerful object detection algorithm that achieves state-of-the-art performance on various benchmark datasets. Its two-stage architecture allows it to accurately detect objects while maintaining computational efficiency.

3.3 YOLO :

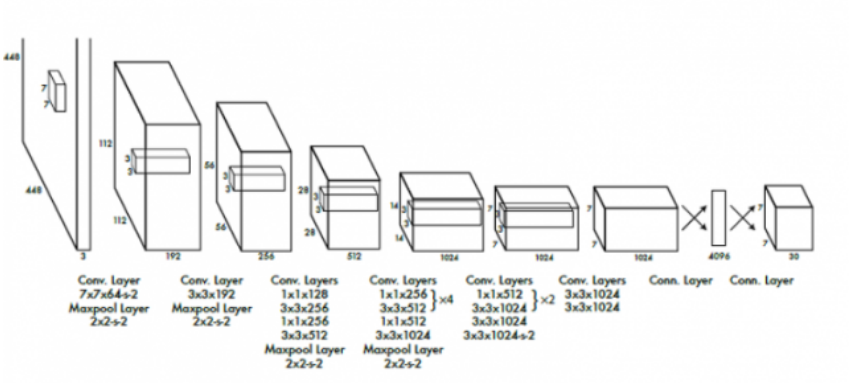

Figure 2: YOLO Architecture (image courtesy: ODSC)

YOLO (You Only Look Once) is a real-time object detection system that uses a single neural network to perform object detection and classification. The architecture of YOLO can be divided into three main parts: the input processing stage, the convolutional neural network (CNN), and the output stage.

- **Input Processing Stage:** The input to YOLO is an image, which is first resized to a fixed size and divided into a grid of cells. Each cell corresponds to a fixed region in the original image. For each cell, YOLO predicts a fixed number of bounding boxes and the class probabilities of the objects that are present in the cell.
- **Convolutional Neural Network:** The CNN of YOLO is composed of 24 convolutional layers followed by 2 fully connected layers. The first 20 convolutional layers use a standard architecture with max pooling and ReLU activation functions. The last 4 convolutional layers use a 1x1 kernel to reduce the feature map size and increase the receptive field.
- **Output Stage:** The output of the CNN is a feature map that encodes the presence and location of objects in the input image. For each cell in the grid, YOLO predicts a fixed

number of bounding boxes and the class probabilities of the objects that are present in the cell. Each bounding box is represented by 5 numbers: the coordinates of the center of the box, the width, and the height. The class probabilities are represented by a vector of scores, where each score corresponds to a specific object class.

During training, YOLO minimizes a loss function that penalizes errors in both the bounding box coordinates and the class probabilities. The loss function is designed to balance the errors in localization and classification so that the network can accurately detect objects in the input image.

Overall, YOLO is a fast and accurate object detection system that can be used in real-time applications such as video surveillance and autonomous driving.

3.4 Transformers:

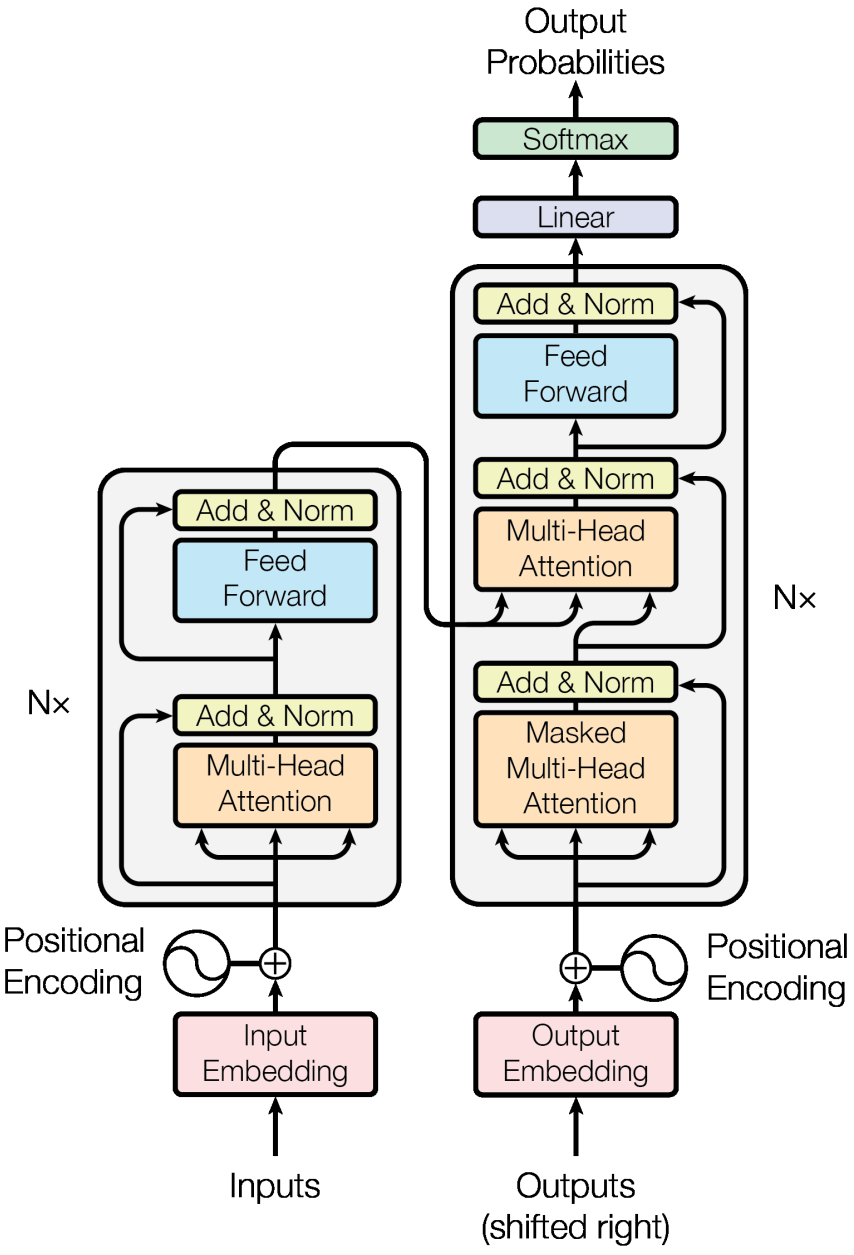

Figure 3: Transformer Architecture (image courtesy: [12])

The Transformer is a type of neural network architecture that has been widely used for various Natural Language Processing (NLP) tasks, such as language modeling, machine translation, and text generation. It was first introduced in the paper "Attention Is All You Need" by Vaswani et al. in 2017 [12], and it has become one of the most popular NLP models due to its excellent performance and flexibility.

The architecture of the Transformer can be divided into two main parts: the encoder and the decoder. The encoder is responsible for processing the input sequence, while the decoder generates the output sequence. Both the encoder and the decoder are composed of multiple layers of self-attention and feedforward neural networks.

The architecture of the transformer can be found in the following:

- **Input Embedding Layer:** The input to the Transformer is a sequence of tokens, usually represented as integers or one-hot vectors. These tokens are first embedded into a continuous vector space using an embedding layer. The embedding layer maps each token to a dense vector of fixed dimensionality, which represents its semantic meaning.

- **Positional Encoding Layer:** Since the Transformer does not have any recurrent or convolutional layers, it needs a way to incorporate the sequence position information. To achieve this, the Transformer uses a positional encoding layer that adds sinusoidal functions of different frequencies to the token embeddings. These sinusoidal functions have a fixed pattern that allows the model to capture the relative positions of the tokens.

- **Encoder:** The encoder consists of a stack of identical layers, each of which has two sublayers: a self-attention layer and a feedforward layer.

  - **Self-Attention Layer:** The self-attention layer computes the attention weights between each token in the sequence and all the other tokens, including itself. This allows the model to weigh the importance of each token based on its relevance to the other tokens. The self-attention layer takes as input the embeddings and positional encodings of all the tokens and outputs a set of context vectors, which are weighted sums of the embeddings.

  - **Feedforward Layer:** The feedforward layer applies a fully connected neural network to each context vector independently. It consists of two linear transformations with a non-linear activation function in between.

- **Decoder:** The decoder is similar to the encoder, but it has an additional sublayer: the masked multi-head attention layer.

  - **Masked Multi-Head Attention Layer:** The masked multi-head attention layer is similar to the self-attention layer in the encoder, but it has a masking mechanism that prevents the decoder from attending to future tokens. This is necessary because the decoder generates the output sequence one token at a time, and it should not have access to the tokens that come after the current token. The masked multi-head attention layer takes as input the embeddings and positional encodings of the output tokens and the context vectors generated by the encoder, and it outputs a set of attended vectors, which are weighted sums of the encoder context vectors.

  - **Self-Attention Layer and Feedforward Layer:** The decoder also has a self-attention layer and a feedforward layer, which are similar to those in the encoder. The self-attention layer attends to the output tokens themselves, allowing the decoder to weigh the importance of each token based on its relevance to the other tokens in the output sequence.

- **Output Layer:** The output layer is a fully connected neural network that maps the decoder output to the final predicted values. For example, in machine translation, it maps the decoder output to a probability distribution over the target vocabulary.

3.5 Vision Transformers (ViT):

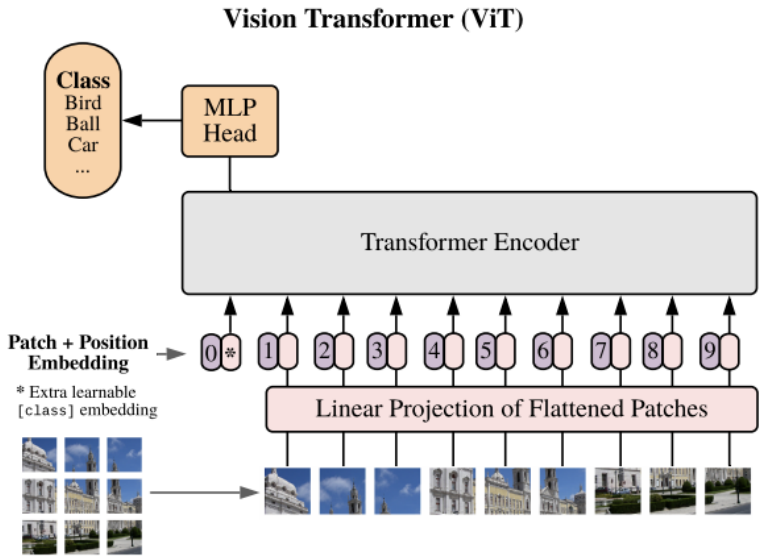

Figure 4: Architecture of Vision Transformers (ViT) (image courtesy: [3])

The Vision Transformer (ViT) is a neural network architecture that applies the self-attention mechanism to image classification tasks. The architecture can be divided into three main parts: the input embedding stage, the transformer encoder stage, and the classification stage.

- **Input Embedding Stage:** The input to ViT is an image, which is first divided into non-overlapping patches of fixed size. Each patch is then linearly projected to a fixed dimension to obtain a patch embedding. Additionally, a learnable class token is added to the sequence of patch embeddings, which represents the image as a whole.
- **Transformer Encoder Stage:** The patch embeddings and class token are fed into a stack of Transformer encoders, which apply multi-head self-attention and feedforward layers to capture the spatial relationships between the patches. Each encoder also has a residual connection and layer normalization.
- **Classification Stage:** The final output of the Transformer encoder stack is fed into a multi-layer perceptron (MLP) with a softmax activation function to produce class probabilities for the input image.

During training, ViT is trained with a cross-entropy loss function that compares the predicted class probabilities with the ground truth labels. The weights of the network are updated using backpropagation and stochastic gradient descent.

Overall, ViT is a powerful architecture for image classification that leverages the self-attention mechanism to capture global spatial relationships between image patches. The architecture has achieved state-of-the-art results on several benchmark datasets, demonstrating its effectiveness in practice.

4 Methods:

4.1 Object detection with CNN backbone:

We used two CNN-based backbone models for Faster R-CNN architecture. First, we extracted 5 feature layers from VGG model to be used as a feature pyramid. These feature maps are then passed

through a Region Proposal Network (RPN) and a Region of Interest (RoI) pooling layer for further processing. Similarly, we use ResNet50, to extract 5 feature layers to be fed into the Faster R-CNN object Detection Head. Selecting feature maps from multiple layers provides information at multiple scales. We also utilize anchors of various sizes (32, 64, and 128) and aspect ratios (0.5, 1, and 2) to generate region proposals.

In the image below we give a schematic of the Faster R-CNN model with a VGG backbone. The one with ResNet50 backbone is the same as the original Faster R-CNN model in 3.2.

[4]

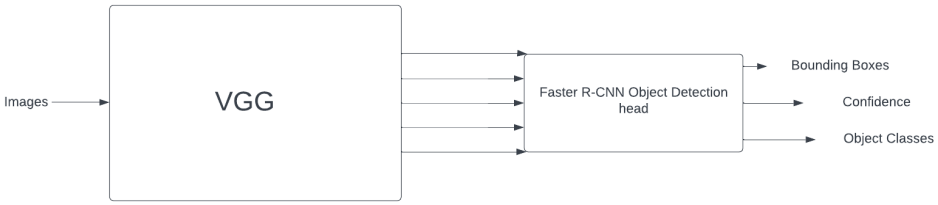

Figure 5: Architecture of Faster R-CNN with VGG Backbone (image courtesy: We !)

4.2 Object detection with Transformer backbone:

ViT is designed to process images of a fixed size, but this is not always possible in practice. To address this issue, we introduce a zero padding layer to ensure that images of varying sizes can be processed by the ViT model. We then extract the features generated by the ViT model just prior to the final layer. These features are then permuted into a 3D feature map with dimensions of 768 x number of patches x number of patches.

In this kind of model, we picked the feature maps from the second last layer of the ViT backbone, and then these 3D feature maps are fed into the Faster R-CNN object detection head to predict objects in the image in a similar manner as described in 4.1.

In the image below we give a schematic of the Faster R-CNN model with ViT backbone.

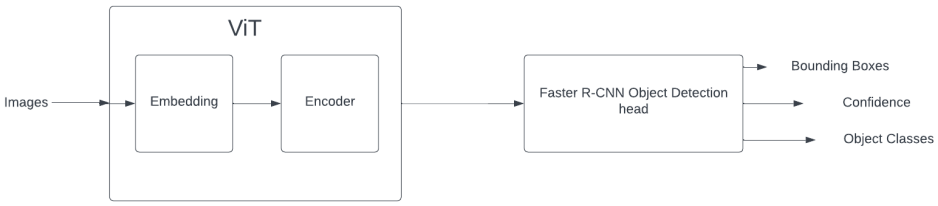

Figure 6: Architecture of Faster R-CNN with ViT Backbone (image courtesy: We !)

4.3 Object detection with End-to-End Transformer:

Feature maps from ViT models are passed through 3 fully connected layers for predicting bounding boxes, confidence, and object classification. The predictions are then compared against the targets using permutation-invariant losses. We also use feature maps from the ResNet50 model to be fed into the end-to-end ViT model.

In the two images below we give schematics of the end-to-end ViT models we implemented.

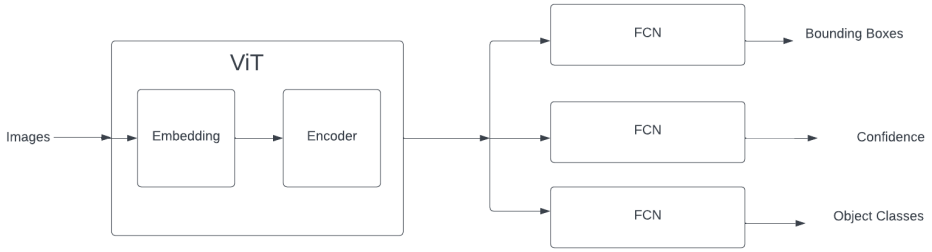

Figure 7: Architecture of End to End ViT (image courtesy: We !)

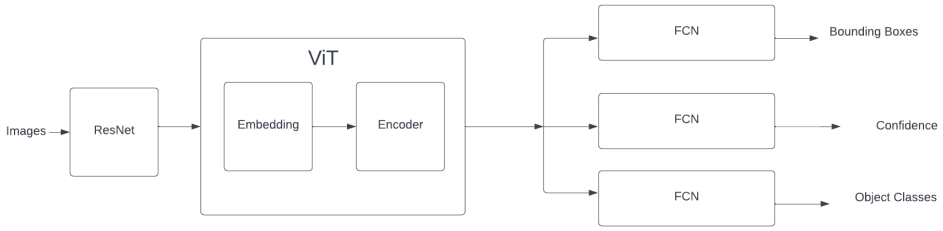

Figure 8: Architecture of End-to-End ViT with ResNet backbone (image courtesy: We !)

4.4 Method of Loss Calculations:

- **Faster RCNN Loss:** Since object detection involves identifying an object accurately and also its location in the image, these models are usually trained on two types of losses. The classification loss(L_{cls}) measures the accuracy of the model in predicting the class or category of the object, while the localization loss(L_{loc}) measures the accuracy in predicting its location in the image. The classification loss is calculated using Cross Entropy Loss and the localization loss is calculated using smooth L_1 and they are combined into a single loss function as $L_{cls} + \lambda L_{loc}$.
- **Permutation Invariant Loss:** This loss is considered when we are implementing end-to-end ViTs. When training an object detection model, the input data consists of an image and a set of ground truth boxes that indicate the location and size of the objects in the image. Since the order in which the objects appear in the ground truth boxes should not affect the performance of the model, a permutation invariant loss is used to train the model to detect the objects in any order. We used the Hungarian matching algorithm for this purpose, which matches the target bounding boxes with the detected bounding boxes based on their IOU scores. To implement Hungarian Matching for object detection, we used the predicted bounding boxes as the rows and the ground truth bounding boxes as the columns of a matrix. The cost or distance between each pair of bounding boxes was computed and stored in the corresponding cell of the matrix. Then, the Hungarian algorithm can be applied to find the optimal assignment of rows to columns that minimizes the total cost or distance. Once the optimal assignment is found, we used it to compute the loss for the object detection task.

4.5 Metrics used to evaluate performance:

For comparing the performance of different models, we use Average Precision(AP) as a common metric to evaluate these models. AP is calculated by first computing the precision and recall for each class of object in the dataset, and then taking the average of the precision values at different recall levels.

Precision is the fraction of the predicted bounding boxes that correctly match with the ground truth bounding boxes. It measures how many of the predicted bounding boxes are true positives (TP) and how many are false positives (FP). Recall is the fraction of ground truth bounding boxes that are detected.

$$Precision = TP/(TP + FP)$$
$$Recall = TP/(TP + FN)$$

The precision and recall values are plotted as a curve, known as the precision-recall curve. The AP is then computed as the area under this curve. The AP score can range from 0 to 1, with higher scores indicating better object detection performance.

AP is often reported at different IoU (Intersection over Union) thresholds, as it allows us to see how well it performs at different levels of overlap. A high AP at a high IOU threshold indicates that the model is good at detecting objects with high precision, while a high AP at a low IOU threshold indicates that the model is good at detecting objects with high recall.

To compute the average precision at different IOU thresholds, the model's predictions are sorted based on their confidence scores. Then, for each IOU threshold, a precision-recall curve is created by considering the predicted bounding boxes with IOU greater than or equal to the threshold as true positives and the rest as false positives. Typically, the IOU thresholds used are 0.5, 0.75, 0.9, and sometimes 0.95. These thresholds represent the minimum amount of overlap required between the predicted and ground truth bounding boxes to consider the detection a true positive.

5 Experiments:

For comparison purposes, we conducted experiments on all the models with the same training parameters. We used stochastic gradient descent for the optimizer with a learning rate of 0.005 and weight decay of 0.0005. To avoid getting stuck in a local minimum, we adjusted the learning rate every 5 iterations with a decay of 0.1. We used Pytorch's `lr_scheduler.StepLR` module for this purpose. We also used Faster R-CNN loss for training Faster R-CNN based object detection models and used permutation-invariant losses for training transformer based models. We trained our models for 14910 iterations.

6 Results:

6.1 Average Precision for each model trained on PASCAL VOC:

In the following table, we enlist the performances of all the models we used with respect to the metrics discussed above. For comparison purposes, we also cite similar results for YOLO taken from [8].

Table 1: Performances of different models

MODEL(s)	AP@ IoU=0.5:0.95	AP@ IoU=0.5	AP@ IoU=0.75	Time/iter
Faster R-CNN with ResNet backbone	1.5	5	0.5	0.7249s
Faster R-CNN with pretrained ResNet backbone (on COCO)	24	55	13.7	1.3114s
Faster R-CNN with ViT backbone	0.3	1.2	0.1	0.2277s
Faster R-CNN with ViT backbone & Multiple Anchors	1.5	4.2	0.5	0.5234s
Faster R-CNN with ViT backbone and Corners Aligned	2.1	6.2	0.8	0.7981s
Faster R-CNN with VGG backbone	0	0	0	0.5541s
End to End ViT	0.1	0	0	0.1212s
End to End ViT with ResNet	0.3	0	0	0.3842s
YOLO Pretrained (on COCO)[8]	63.4	49.7	24.3	———-

6.2 Sample images from best CNN and best ViT Models:

In the following, we provide object detection results on two random images from the test dataset for the best models as above. The first image is from the Faster R-CNN with Resnet model. The second image is from the Faster R-CNN with ViT backbone where multiple anchors are used and corners are aligned.

(a) Faster R-CNN with Resnet backbone

(b) Faster RCNN with ViT backbone

Figure 9: Sample image result for object detection

6.3 Loss curves for the best model from above:

We see from the last table that the best model (in terms of 'average precision @ IoU=0.5:0.95') is the 'Faster R-CNN with ViT backbone and Corners Aligned'. So we provide the training and validation loss plots for it, in the following.

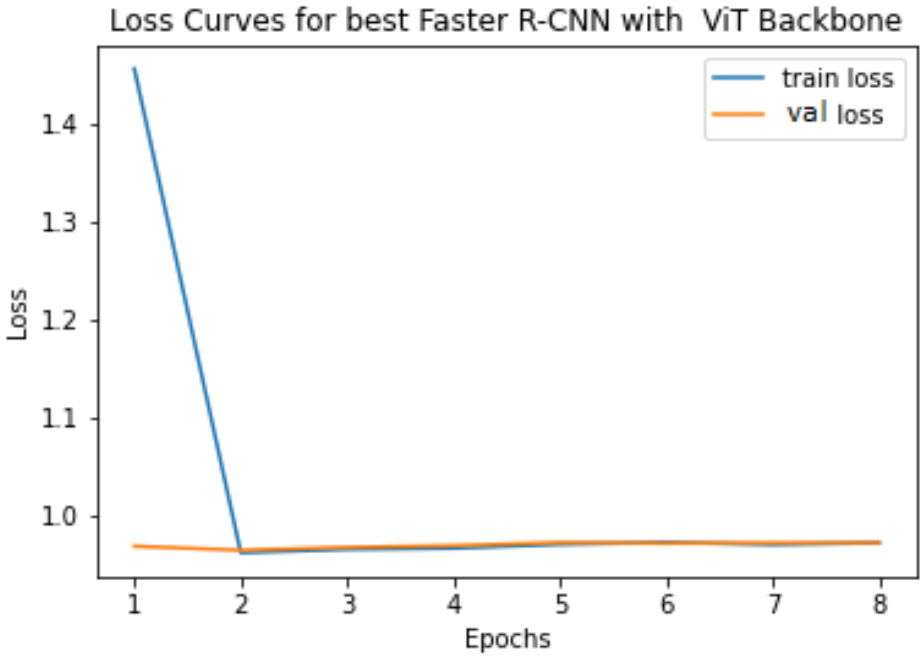

Figure 10: Loss curves for Faster R-CNN model with ViT backbone with Multiple anchors and corners aligned

7 Discussion of Results:

7.1 Anchors:

In object detection using region-based convolutional neural networks, anchor boxes are pre-defined bounding boxes of different sizes and aspect ratios used as reference points to generate proposals of varying scales and shapes. During training, the model learns to adjust the anchor boxes to better align with the ground truth boxes for the object of interest. The number, size, and overlap of the anchor boxes are crucial in achieving high accuracy. Anchor boxes have proven to be a critical component in achieving state-of-the-art performance in object detection. We find that with the increase in sizes of the anchors and aspect ratio, the average precision improves for ViT backbone. Because this allows the Fast R-CNN object detection models to look at multiple different scales simultaneously.

7.2 Interpolation:

ViT models require images of fixed sizes. However, the input images are not of fixed sizes. Hence we will have to either do a bilinear interpolation of the images or pad these images with zeros. We found that interpolating these images provides better results in comparison to padding these images. This could be due to the fact that zero-padding essentially new information into the images.

7.3 Transfer Learning based models:

PASCAL VOC 2007 is a very small dataset with only 2500 training images. This dataset is too small to learn a complex problem such as object detection. Hence training our backbone models on a significantly larger dataset like COCO helps in learning the object detection task from PASCAL VOC better.

7.4 End to End ViT vs Object Detection Head:

It can be seen that end-to-end ViT models performed worse than the Faster R-CNN based models. This is because end-to-end object detection models are forced to learn the structure of the problem on a significantly smaller dataset. However Faster R-CNN models predict anchors, do an RoI pooling of the anchors which aids them in learning the structure of the problem significantly faster.

7.5 Visualization of loss curves :

The training loss curve goes steeply down and then become constant, matching then with the validation loss. From here we can say that the attention module of the 'Faster R-CNN with ViT backbone and Corners Aligned' learns the features very quickly. This shows that attention is a quite fast and powerful method for learning features.

8 Conclusions:

We have put multiple Object Detection Head-algorithms with CNN-based backbone models and transformer-based models. We also compared these models with end-to-end transformer-based models. We found that ViT-based models performed better than CNN-based models and which in turn performed better than end-to-end ViT models. These ViT based models are extremely fast to train hence it can be understood that using a purely attention-based model for the purpose of object detection is fruitful given that we have a large enough training dataset. We would like to extend this work by training the models on COCO dataset. We would want to pre-train our ViT backbone models on COCO dataset. We would want to extend ViT model by adding an decoder model for training an End to End ViT model for object detection.

References

[1] Nicolas Carion, Francisco Massa, Alexander Kirillov, Ross Girshick, Li Fei-Fei, and Trevor Darrell. End-to-end object detection with transformers. In *ECCV*, 2020.

[2] Zhuangbiao Chen, Yulin Yang, Ziyan Huang, Ming Zhang, Jianfeng Guo, Junzhou Huang, and Jiebo Xu. Visual transformer for object detection. In *CVPR*, 2021.

[3] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale, 2021.

[4] Ross B. Girshick. Fast R-CNN. *CoRR*, abs/1504.08083, 2015.

[5] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016.

[6] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. Ssd: Single shot multibox detector. In *ECCV*, 2016.

[7] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *CVPR*, 2016.

[8] Joseph Redmon, Santosh Kumar Divvala, Ross B. Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. *CoRR*, abs/1506.02640, 2015.

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

[10] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *arXiv preprint arXiv:1409.1556*, 2014.

[11] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *CVPR*, 2015.

[12] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *CoRR*, abs/1706.03762, 2017.

[13] Sixiao Zheng, Jiacheng Lu, Hengshuang Zhao, Xiatian Zhu, Zekun Luo, Yabiao Wang, Yanwei Fu, Jianfeng Feng, Tao Xiang, Philip HS Torr, et al. Rethinking semantic segmentation from a sequence-to-sequence perspective with transformers. In *ECCV*, 2020.

Team Contribution:

The project is a result of a team effort where all of us contributed as a group. Both ran the codes and provided the necessary data for plotting and making tables in this report. However, roughly the contribution of each team member is listed below:

- Nishanth was involved in implementing the codes for all the CNN and ViT-based models. In the report, he was responsible for the sections 'Discussion of Results, Conclusions'. He also led the team in conducting appropriate experiments, supervised the report writing, and finally, he presented our work in front of the class.
- Soumya primarily worked on implementing the codes for Faster RCNN with ViT, VGG, and end-to-end ViT along with Nishanth and also trained the models. In the report, he helped in writing the sections of 'Introduction, Background Material, Results, Conclusion' along with the subsection of 'Permutation Invariant Loss'.
- Pranav worked on the Fast RCNN part with VGG backbone with Nishanth and ran experiments on a few other models to check their efficiency and to evaluate which models are working well for our use case. In the report, Pranav worked on the Abstract and Related Work sections of the report.
- Daamini worked on implementing the DataLoader module along with Nishanth, calculating the IOU and mAP metrics for evaluation. She helped in writing Loss calculations and Metrics subsections in the Methods section.