

VISION TRANSFORMERS

1 Abstract

In this project, the Transformer architecture is applied to detect and pinpoint objects. The dataset used is Caltech 101 from the Caltech library. Following the research published in 2021 “An Image is Worth 16*16 Words”, this project will implement a similar architecture to detect objects. Using the self-attention mechanism on patches of images is a way to derive attention maps which help focus on what is relevant in the image.

2 Introduction

Transformers were widely used for NLP tasks and their success relates to the use of a Self-Attention mechanism that looks at the input sequence and gives importance to certain words, which helps the decoder to focus on context that is relevant. The idea of attention for computer vision was first proposed by Larochelle and Hinton - by looking at different parts of the image, we can collect information about the shape of the object and classify it accordingly. Attention is either used in concurrence with CNN or serves as a proxy for some features of CNN. A pure transformer is tested in the research “An Image is Worth 16*16 Words”, and it shows that it can work extraordinarily well for computer vision tasks.

3 Methodology

Transformers are devoid of the translation invariance and locally restricted receptive field (such as in CNN). Transformers can model long-range dependencies and multi-headed attention is the secret behind the transformer’s success. Each attention head has 3 tensors called Query, Key and Value. The output of the multi-headed attention block is given by

$$Attention(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

The attention value from element i to j is obtained by the dot product of Q_i and K_j which measures similarity. The QK^T term computes this for every element.

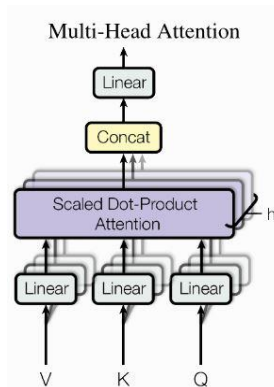


Figure 1: Multi-Headed Attention Block

The $\frac{1}{\sqrt{d_k}}$ is a scaling factor and it helps maintain variance of the attention values and ensure that the SoftMax term does not saturate.

One attention block may not be able to attend to the whole image. Since there are multiple aspects in an image, multiple heads of these attention blocks are generally used and the outputs of multiple heads are concatenated.

The Vision Transformer architecture as shown in the image has 4-5 stages. The first step is to split the images into different patches as the Transformer works on sequences. The Attention mechanism is used for patches related to the image, i.e., Attention weights are calculated for every pair of features, and each feature contains information about other features.

Positional embedding is used to mark patches relative to the overall image.



Figure 2: Image split into patches

The sequence of images (tokens) along with the positional embedding is fed into the Transformer Encoder which contains layers of Normalization, Multi-headed attention blocks and Multi-Layer Perceptrons (MLP). Normalization helps in adapting to the variation in images. Attention block is used to generate attention maps from the embedded tokens which help focus on what is relevant in the image. Finally, the MLP outputs four dimensions that represent the bounding box to detect the coordinates of objects.

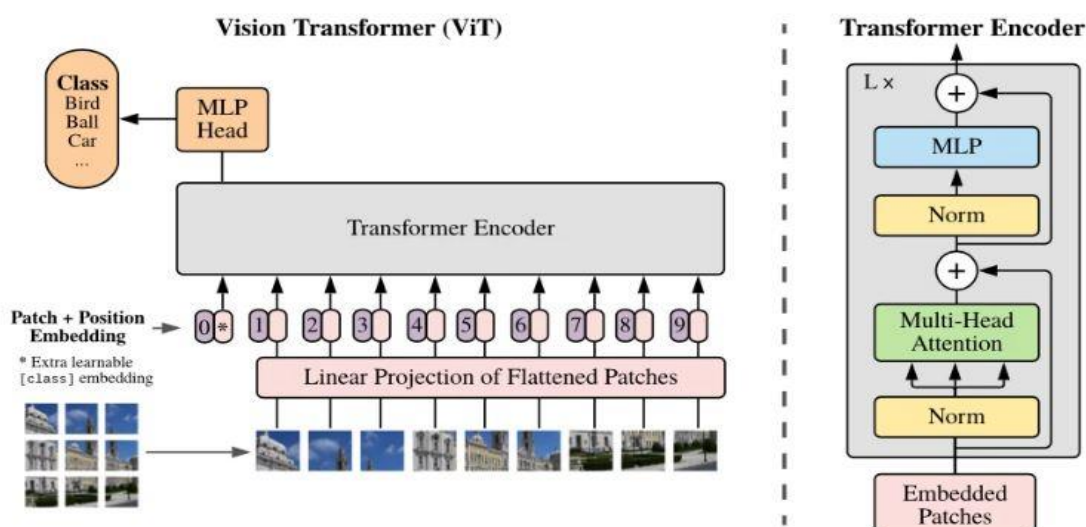


Figure 3: ViT Architecture

3.1 Metric:

Intersection over Union is the metric used for this project. It is calculated as a ratio of the Intersection of the predicted and ground-truth bounding box with the Union of the predicted and actual bounding box.

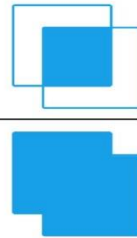
$$IoU = \frac{\text{Area of Overlap}}{\text{Area of Union}}$$


Figure 4: IoU metric calculation

As shown below, an IoU value of 0.6 or greater can be considered as reasonable, and greater than 0.7 can be deemed excellent.

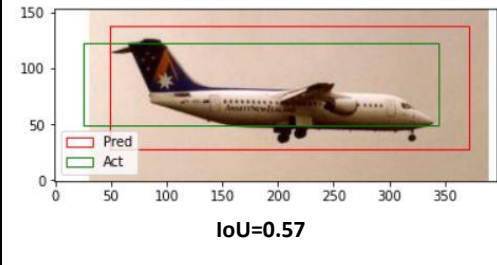
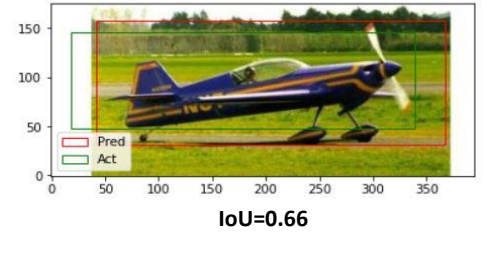
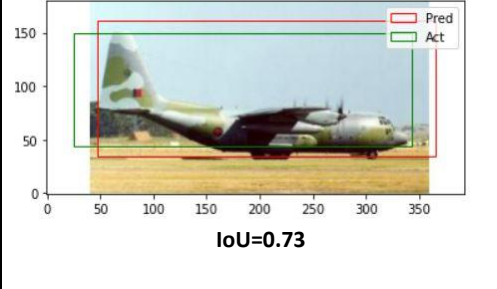
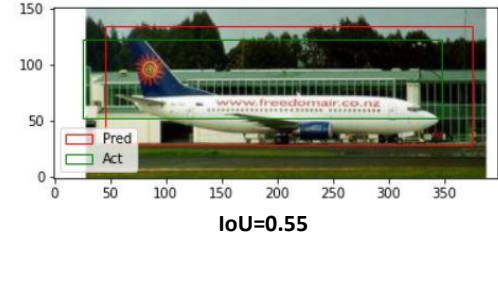
 <p>IoU=0.57</p>	 <p>IoU=0.66</p>
 <p>IoU=0.73</p>	 <p>IoU=0.55</p>

Table 1: IoU values and Bounding Box representation

3.2 Dataset

The Caltech 101 dataset contains pictures of objects in 101 categories. Each category has a minimum of 40 images and some categories have up to 800 images. The size of each image is around 300x200 pixels. Each image has its own annotation which tells us the coordinates of the bounding box.

4 Challenges

Major challenges encountered while working on the project were:

Resource constraints: Working on 2 GB worth of images was a challenge. It was difficult to run models with the entire dataset on a personal computer. Manual data generators were deployed unsuccessfully due to restrictions from Tensorflow eager execution. Google Colab on the other hand was immensely slow in processing and loading images from the drive. Both resources

were utilized to speed up the process - a personal computer for loading and processing the dataset and Google Colab for running the model.

OOP for intermediate steps: Splitting the images into patches and feeding them into the network required building sub classes of the Keras Layers. This required a deeper understanding of class inheritance in Python

Custom Metric: Creating a custom metric was a challenge because it had to comply with the Tensorflow eager execution. Adding the “@tf.function” helped overcome this issue

Pre-trained models: As mentioned in the research “An Image is Worth 16*16 Words”, Vision transformers work most effectively when pre-trained on large datasets and fine-tuned to smaller ones. Transformers lack some of the inductive biases to CNNs, such as translation equivariance and locality, and therefore do not generalize well when trained on insufficient amounts of data. However, Keras did not have the functionality to load the pre-trained model with a customizable head

Visualizing Attention blocks: Visualizing attention blocks could give us vital information if the model is learning the right features. For example, the Key in the multi-head attention could highlight the outline of the image. When dot product is performed between the Query and the Key, the Key basically tells the entire image on where the aeroplane is (assuming Query just highlights the entire image). Visualization was attempted but not performed due to its complexity and it required more time

Image Augmentation: Image augmentation would help in generalizing the model. A vanilla augmentation using Keras packages cannot be performed as this is not a classification task. The bounding box needs to be changed accordingly. This task was also attempted but not implemented because of its complexity

5 Network Architecture

Minor changes in the “An Image is Worth 16*16 Words” architecture was implemented and tested based on results obtained. Most changes were made keeping resource constraints in mind.

- Each image was split into 36 patches as it converged faster when compared to 81 patches. Images were normalized before feeding into the network
- The number of heads in the Multi-Headed Attention block was changed to 3 to reduce the complexity of the network. It was tested against 4 heads and heads=3 performed better
- MLP block in the encoder consisted of 2 Dense layers with ReLU activation and no regularization. The size of the Dense layers was maintained so that residual connection could flow through after the MLP block. Regularization was tested but not used
- The MLP head outside the encoder consisted of 4 dense layers (excluding the output layer) with ReLU activation. Increasing beyond 4 layers did not give any improvement in the results. Regularization was tested but not used as it did not improve the results. The output layer was also a dense layer with 4 neurons without activation as we required prediction of the coordinates of the bounding box. Dropout layers were tested in between the Dense layers and it did not help in improving the results

- Adam optimizer with learning rate = 0.001 produced the best results. It was compared against Stochastic Gradient Descent (SGD) and Nesterov Accelerated Gradient (NAG)
- Loss function used was Mean Squared Error as we had a regression problem at hand in predicting the bounding box coordinates
- Metric used was Intersection Over Union (IoU) for the bounding boxes
- Batch size used was 64 so that the network ran fast enough without causing memory issues
- Network was trained for 15-20 epochs for most configurations as the validation and training IOU plateaued after 3-5 epochs (finally, epochs=3 was decided for the best configuration)
- CalTech101 dataset has 101 categories of objects. Although we have an object detection problem at hand, class distribution among Training, Validation and Test sets are important so that the model learns to build the bounding box on all objects. Sklearn's train_test_split module was used to ensure class distribution is maintained

Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	[(None, 144, 144, 3)]	0	[]
patch_process_1 (PatchProcess)	(None, 36, 128)	225920	['input_1[0][0]']
layer_normalization (LayerNormalization)	(None, 36, 128)	256	['patch_process_1[0][0]']
multi_head_attention (MultiHeadAttention)	(None, 36, 128)	197888	['layer_normalization[0][0]', 'layer_normalization[0][0]']
add (Add)	(None, 36, 128)	0	['patch_process_1[0][0]', 'multi_head_attention[0][0]']
dense_1 (Dense)	(None, 36, 128)	16512	['add[0][0]']
dense_2 (Dense)	(None, 36, 128)	16512	['dense_1[0][0]']
add_1 (Add)	(None, 36, 128)	0	['add[0][0]', 'dense_2[0][0]']
layer_normalization_1 (LayerNormalization)	(None, 36, 128)	256	['add_1[0][0]']
flatten (Flatten)	(None, 4608)	0	['layer_normalization_1[0][0]']
dropout (Dropout)	(None, 4608)	0	['flatten[0][0]']
dense_3 (Dense)	(None, 512)	2359808	['dropout[0][0]']
dense_4 (Dense)	(None, 256)	131328	['dense_3[0][0]']
dense_5 (Dense)	(None, 128)	32896	['dense_4[0][0]']
dense_6 (Dense)	(None, 32)	4128	['dense_5[0][0]']
dense_7 (Dense)	(None, 4)	132	['dense_6[0][0]']
=====			
Total params: 2,985,636			
Trainable params: 2,985,636			
Non-trainable params: 0			

Figure 5: Network Architecture and parameters

5.1 Results and Discussion:

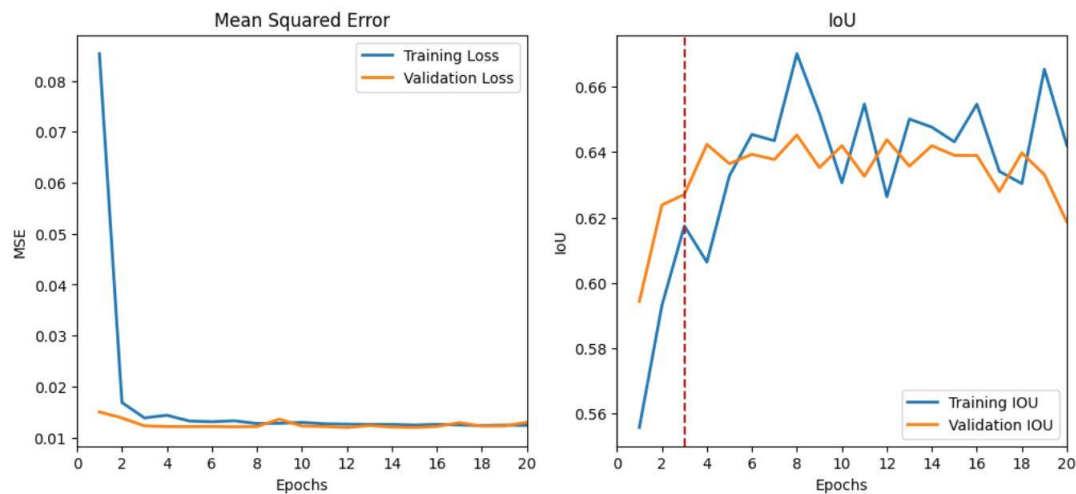
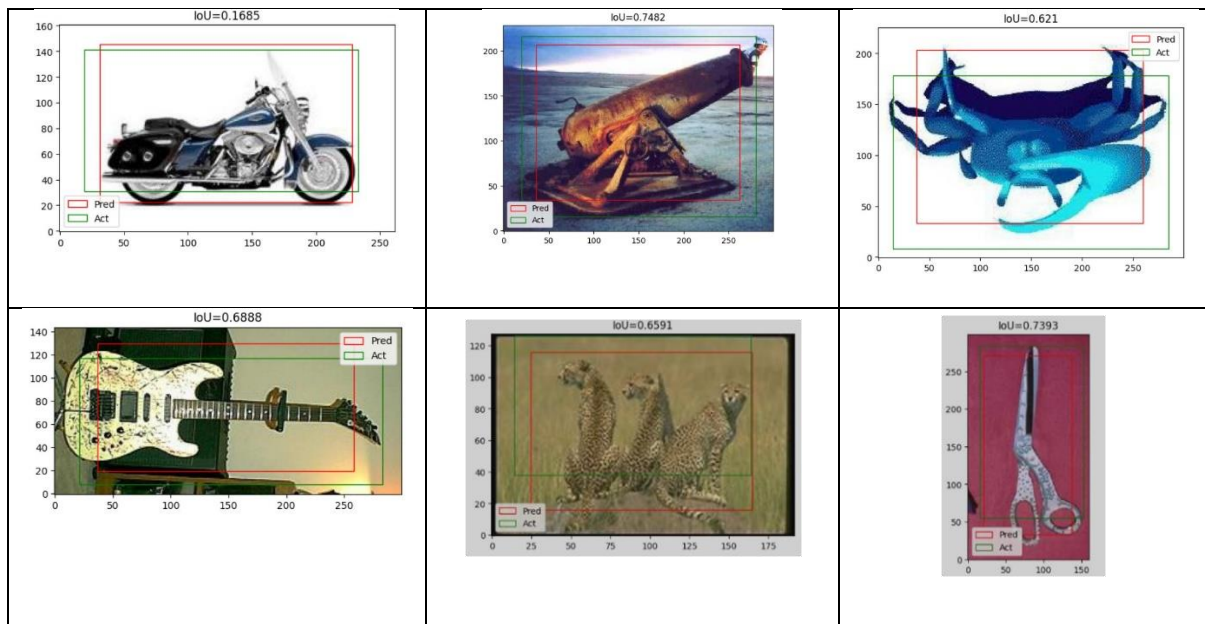


Figure 6: Loss and IoU by Epochs

The model was trained for 20 epochs. As shown above, the training can be stopped at epochs = 3 as the validation IoU starts to plateau post that. Another reason for stopping at epochs=3 is that model tends to predict constant values after a certain point.

The training and validation IoU are not very far apart and for this reason not much regularization was applied to the network. Increasing the complexity of the network did not increase the training IoU by much.

Here are the predictions for a few images:



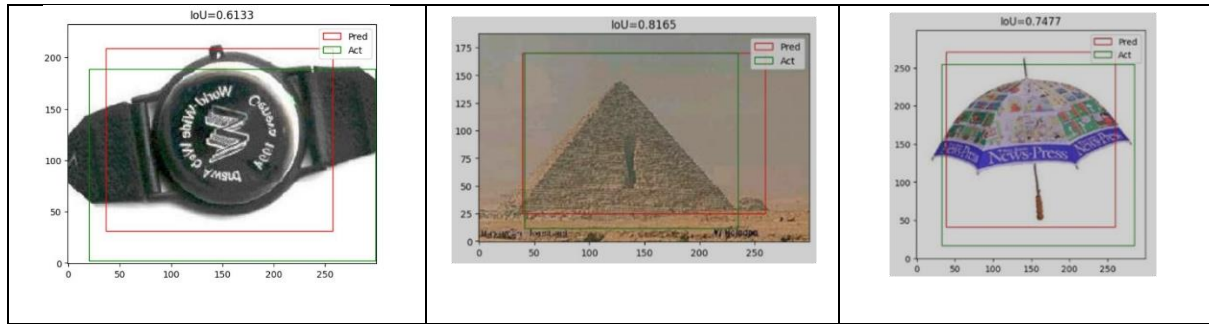


Table 2: Predicted Bounding Boxes

The below table shows the IoU for a few categories:

Category	IoU		Category	IoU
Motorbikes	0.77		Watch	0.60
Crab	0.67		Hedgehog	0.76
Anchor	0.44		Starfish	0.55
Lotus	0.57		Soccer Ball	0.69
Elephant	0.71		Dolphin	0.53

Table 3: IoU by Category

5.2 Test Results

The final model was trained with all the data (training and validation) with epochs=6 and then it was tested against the test set using the below configurations as it gave the best results (Note at epochs = 6, there is not much overfitting):

Optimizer: Adam with learning rate = 0.001

Epochs: 8

Regularization: None

Loss Metric: Mean Squared Error

Batch Size: 64

Here are the results:

Training IoU: 0.6174

Validation IoU: 0.6270

Test IoU: 0.6508

Test IoU is pretty close to the training and validation IoU which suggests that the model works well. Test IoU is slightly higher as we are training with both training and validation sets.

6 References

- Dosovitskiy, A. (2020, October 22). An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. arXiv.org.
- Vaswani, A. (2017, June 12). Attention Is All You Need. arXiv.org.
- Yang, M. (2022, June 1). Visual Transformer for Object Detection. arXiv.org.
- Caltech 101. (n.d.). <https://data.caltech.edu/records/mzrjq-6wc02>.
- Team, K. (2022, March 27). Keras documentation: Object detection with Vision Transformers. https://keras.io/examples/vision/object_detection_using_vision_transformer/.
- Exploring Explainability for Vision Transformers. (2020, December 31). Jacob Gildenblat. <https://jacobgil.github.io/deeplearning/vision-transformer-explainability>.
- Adaloglou, N. (2021, January 28). How the Vision Transformer (ViT) works in 10 minutes: an image is worth 16x16 words. AI Summer. <https://theaisummer.com/vision-transformer/>.
- Hedu AI. (2020, December 8). Visual Guide to Transformer Neural Networks - Multi-Head & Self-Attention. YouTube.
- Amirhossein Kazemnejad. (2019, September 20). Transformer Architecture: The Positional Encoding. https://kazemnejad.com/blog/transformer_architecture_positional_encoding/.
- Rosebrock, A. (2022, April 30). Intersection over Union (IoU) for object detection. PyImageSearch. <https://pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/>
- Chollet, F. & others, 2015. Keras. <https://github.com/fchollet/keras>
- Abadi, Marten; Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., ... others. (2016). Tensorflow: A system for large-scale machine learning. In 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16) (pp. 265–283).

7 Instructions

Open the cs577_project_final.ipynb file. Set the directory under ‘Update Directory and Settings’ section. Run the sections – ‘Import Libraries’, ‘Update Directory and Settings’, ‘PatchProcess Class’ and ‘ViT Class’.

Proceed to the ‘----Test----’ section and run the blocks individually to test each of the functions created in the ViT Class. If only the final weights are to be evaluated, proceed to the last cell block in ‘----Test----’ section and run it.

The appendix section contains additional codes for further testing if needed.