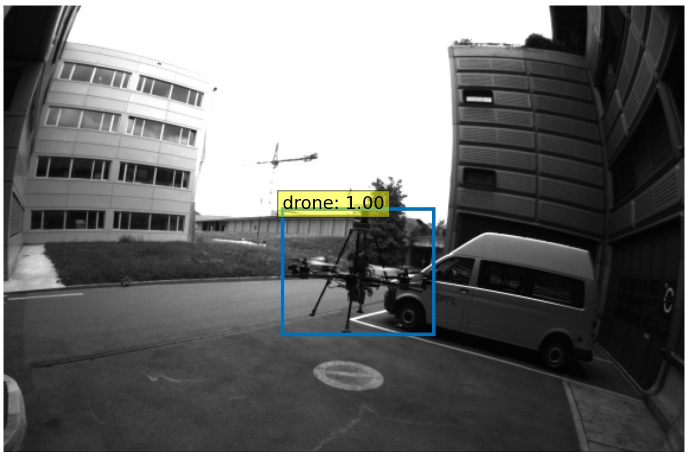
Machine learning CS-433

Drone and pigeon detection



Authors:
Boris David
Silvio Müller
Omar Raita

Supervisors:
Fabian Schilling
Davide Zambrano
Laboratory of Intelligent Systems (*lis.epfl.ch*)

Abstract—**Drone detection is an important step to make drones more autonomous. Convolutional Neural Network (CNN) are a fundamental Machine Learning tool that can be used to create detection algorithms. This report will use the DEtection TRansformer[1] (DETR) architecture developed by Facebook in 2020 to build models able to identify and locate drones and pigeons on images. This architecture will be combined with "ResNet18" and "ResNet50" backbones to train models on images containing respectively drones and pigeons. The general findings of this paper shows promising baseline models that could be even more improved.**

I. Introduction

Nowadays, drone usage has become more and more common in lots of different fields. These can include military operation, express shipping and delivery, aerial photography or recreational uses. The evolution of technologies in Machine Learning has opened more opportunities and ideas to improve the functioning and possibilities for drones.
In a goal to make drones more autonomous, the detection of flying objects using moving cameras is a central step of the process. The area of object detection in Machine Learning is a field that evolves perpetually, new discoveries and algorithms are found each month. In particular, the Detection Transformer (DETR) architecture developed by Carion et al.[1] in 2020 is an adequate tool to produce this object detection algorithm and provides a solid baseline in our case.
The main objective of this project is to develop an algorithm that can detect drones and birds in images taken by moving cameras. This algorithm should be able to be run in real-time on-board a drone, for drone detection. The main machine learning technique that will be used here is model training using neural networks. In this context, the DETR architecture will have to be simplified in order to produce a model that can efficiently be evaluated on images in real-time (on a goal of 10 FPS). This report discusses the findings of the best backbone model to be used on DETR in our problem and describe the challenges involved in training such an algorithm.

II. Data and Methods

A. Data description

The LIS D3 drone detection data set contains a total of 11,822 black and white pictures of resolution 720×540 coming from 18 videos. These frames are divided as described in the following tab.

Video folder	Video 1	Video 2	Video 3	Total :
workshop_flight	568	676	785	2,029
dronedome_flight	833	913	1,058	2,804
passage_flight	763	872	594	2,229
parking_flight	469	679	783	1,931
football_low_flight	440	464	445	1,349
football_high_flight	497	525	458	1,480
Total frames				11,822

The videos are taken from a fixed camera and consist in drone movements at different positions and in different backgrounds.

A second data set is given for this project by the lab : the pigeon data set. This one carries photos taken from a fixed camera where we can mainly find different numbers of pigeons. The images from this set has a resolution of 2592×1520. The data set is already pre-split into a training and a testing set with a total of 2506 frames in the training set and 184 images in the testing set.
For both data sets, each image is accompanied by an annotation file containing the position of the drone in the associated image. This position is described through a bounding box, defined to be a rectangle as small as possible containing the drone. The positions are described using four features : x_{min}, y_{min} which are the coordinates (pixels) of the top-left corner, *width* and *height* of the rectangle. These features can be seen as continuous as they describe the position of an object in a two-dimensional space. These are also features that we want the model to predict, taking as input new images containing drones, that will be converted through the model into PyTorch tensors.

B. Data pre-processing

The data is prepared to be in the COCO format as explained by [2], this is the format that the DETR architecture expects.

C. Splitting the data into training and test sets (drones)

The main aim of this step is to ensure a good generalization. To that extent, we require the test set to be exhaustive and representative for real world application. Two main strategies have been developed:

1) Select some entire videos to constitute the test set. The model would not have prior experience with this set leading to a good generalization test. However, the test set may not be diversified enough in this case, we have no idea of how the model would react in many cases.
2) Select some frames subsets from each video to build the test set. This would allow us to verify how the model reacts in various situations, even if it has already experienced the same backgrounds in the training set.

The decision has been made to focus on the first method and in addition to produce a validation test.
The validation test is the entire video "*workshop_flight_3*" : the environment of this video should be quite familiar for the model since it has been trained on the first two videos taken in the workshop environment.
The test set, however, will take the set of the three videos taken in the passage : "*passage_flight_1*","*passage_flight_2*" and "*passage_flight_3*". The backgrounds of these three videos are sufficiently different from those from the train set. This gives useful results about the capacity of generalization of the model. Finally, the test set, the validation set and the training set contain respectively 2,696, 785 and 9,128 frames for a proportion of about 18.9%/6.6%/74.5%.

D. Literature Review

In the past years the structure of object detection models have become very complex an example for such a model is the Faster R-CNN model developed by Ren et al[3]. Such a model has many hyperparameters that need to be tuned.

In contrary the DETR architecture, developed by carion et. al [1], has a simple structure. This is achieved by using an transformer-model architecture first developed for sequence transaction in natural language processing, which was developed by [4]. The transformer-model architecture has been a revolution in the field of natural language processing because it didn't use recurrent neural networks but rather used an approach fully based on attention mechanisms. Now a short explanation is given on how this architecture has been applied to object detection.

Overview DETR Architecture: The object detection with DETR will be described from left to right referencing to figure 1. First features of the images are extracted by using a convolution neural network (CNN).

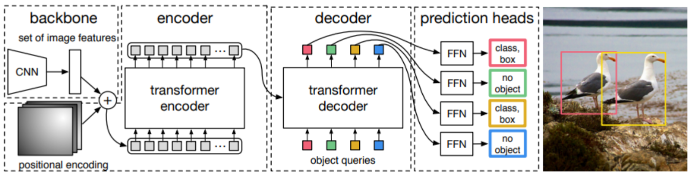


Figure 1. DETR structure [1]

In DETR ResNet CNN's are used. This section of the model is called the backbone. The output of the backbone has the shape $\mathbb{R}^{C\times H\times W}$, typical values for $C = 2048$ and $H, W = \frac{H_0}{32}, \frac{W_0}{32}$. Where H_0 and W_0 are the original dimensions of the image. Since transformers have been developed for sequential data, the image must be flattened before it can be fed into the transformer/encoder. To prevent the loss of location information a positional encoding is added to the output of the backbone. This vector is then passed into the transformer/encoder. The internal structure of the transformer will not be described as this surpasses the scope of this project. The encoded features are then passed on to the transformer decoder as a side input. The main input of the transformer/decoder are the object queries. Since it is a transformer we need the same number of object queries as the number of object we want to detect. They are randomly initialized and then improved during training.

Finally, the output of the transformer/decoder gets passes to a feed forward network, which makes either a class prediction or a bounding box prediction. Each output of the transformer/decoder corresponds to one class and bounding box prediction.

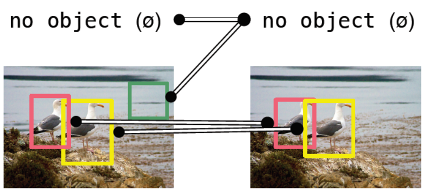


Figure 2. Calculation of the cost with bipartite matching [1]

To calculate the loss a bipartite matching is done. The predictions are matched one to one with one of the ground truth bounding boxes. This is done while reducing the total differences between the matched pairs. The Hungarian algorithm is used for this. If there are more queries than ground truth the rest of the predicted boxes are matched to "no object", this can be seen in figure 2. After the matching, the overall loss function can be calculated which is a linear combination of the l1 loss and GIoU loss.

E. Hyperparameters

Now we want to explore the possible hyperparameters that we could adjust. Since transformers take a very long time to train and our computation resources are limited we have to very be cautious about the choice of hyperparameters we want to evaluate.

The most obvious parameter to adjust is the number of queries in the detector. The number of queries determines how many objects can be detected in one image. Since for our application on drone and pigeon detection it's unlikely that more than 10 objects are in a frame, the number of queries could be reduce. This could speed up the evaluation. The inference times for DETR with different number of queries has been evaluated, these results are reported in figure 3. The inference time stays approximately constant. Probably because the queries in the decoder are calculated in parallel.

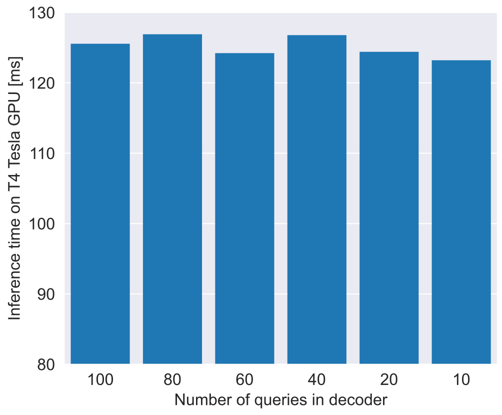


Figure 3. Inference time for different queries

Depending on whether a different backbone than a ResNet is used, the hyperparameters of this new backbone has to be optimized. Especially the learning rate. In the DETR paper the suggest taking a learning rate an order smaller in the backbone than in the transformer.

F. Embedded Hardware

The algorithm for drone detection needs to perform object detection on board the flying drone in real-time using embedded hardware. It is therefore important to be able to assess and compare models with different backbone architectures directly on the hardware that is used for the object detection.

In order to do so, we directly evaluate the models on the embedded hardware using an image set from the coco data set.

The embedded hardware is a Jetson TX2. It is built around an NVIDIA Pascal family GPU and is specially designed for fast and power-efficient embedded AI computing [5].

G. *Choice of Backbone*

As we have mentioned, the DETR architecture requires a backbone that extracts features from the images, which are fed into the transformer. DETR has been designed to use backbones specified in the "torchvision.models" class [1]. It has also been tried to modify the core of the DETR code in order to implement a supplementary backbone (EfficientNet). One of the main selection criterion was the evaluation time: as our goal is to make the algorithm run in real-time on the Jetson in the case of the drone data set. This is not the case for the pigeon data set. The inference time of the different components of DETR were analysed. It can be seen from figure 6 that the inference of the transformer already takes more than 200ms. But for real time we need an total evaluation time less than 100ms. The only way how the transformer time is reduced without modifying its structure, is to reduce the size of the input. "ResNet50" and "ResNet101" both have 2048 channels as an output. That's the reason the inference times are the same for the transformer. Therefore, a backbone with fewer channels has to be taken, for example "ResNet18", which has 512 channels.

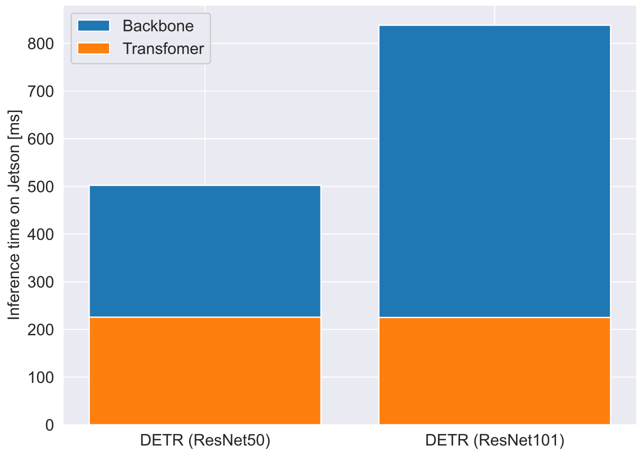


Figure 4. Inference time of the different components in DETR

Various backbones were tested by image classifying 100 coco images and taking the median of the evaluation time. These results are shown in figure 5 on the abscissa and the Top-1 Accuracy on the ordinate. The evaluation time on the Jetson with a "ResNet50" takes 500ms which is too long for real-time. A faster backbone has to be used, this is indicated with the vertical blue line. Therefore, a "MobileNet" or a "ResNet18" could be used, which is in accordance with the statement of the last paragraph. For the drone data set we choose the "ResNet18" as it was a lot simpler to implement and for the pigeons we used "ResNet50". We chose "ResNet50" because there is already a pre-trained DETR model for it that has been trained on coco and has a good trade-off between evaluation time and accuracy.

[1]https://pytorch.org/docs/stable/torchvision/models.html

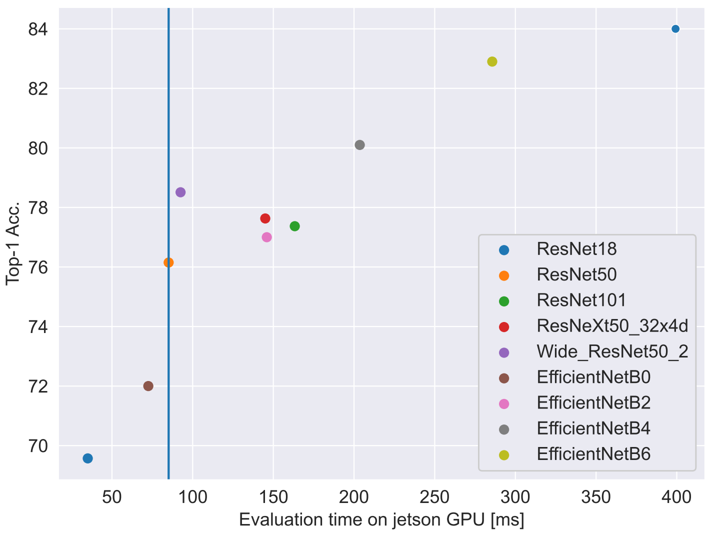


Figure 5. Plot of the inference time vs the Top-1 accuracy taken from the research papers: ResNet[6], ResNeXt[7], Wide ResNet[8], MobileNet[9], EfficientNet[10]

H. *Sanity Check*

An important step to verify the viability of the model is to perform a sanity check. In our case, it consists in training the chosen model on a training and testing sets constituted with one single same image. We expect, after a few epochs, an over-fit of the model. As we have only one image to train on, the model should perfectly predict the bounding boxes of this image. To that extent, we trained the DETR model with pre-trained "ResNet18" backbone for 300 epochs and got the following error plots in the drone case.

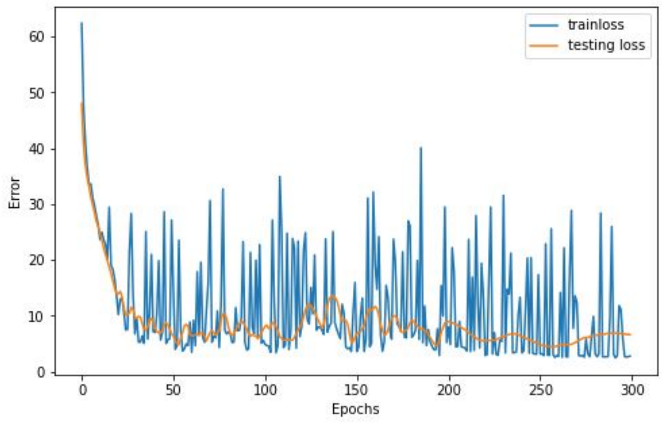


Figure 6. Training and testing loss of the sanity check (drones)

We see that the training plot in particular seems quite volatile and the error stagnates at around 8-9. A similar plot for the pigeon data set is produced under the same conditions (but under "ResNet50" instead of "ResNet18"). This discourages us to train the whole models on the full data.

III. Results

As already discussed in the last section, the sanity checks were not encouraging, we decided against a training on the full data sets. So we focused more on the pre-trained models and fine-tuning those models on our data sets.

A. Baseline Pigeons

Since the coco already has a class "birds" we used the pre-trained models given by the DETR research group to evaluate the pigeon data set. The results are collected in the following table III.

Average Precision	IoU=0.50:0.95	maxDets=100	0.245
Average Precision	IoU=0.50	maxDets=100	0.504
Average Recall	IoU=0.50:0.95	maxDets=10	0.472
Average Recall	IoU=0.50:0.95	maxDets=100	0.522

Table I
Solutions of pre-trained DETR with ResNet50 on pigeons data set

These results are better than the one observed with the Faster RCNN with inception v2 backbone pre-trained on the coco data set (AP@[IoU=0.5:0.95]= 0.0915). Therefore, it is expected, that DETR could give very good results if trained or fine-tuned on the pigeons data set.

B. Best Model Pigeons

For the pigeon data set we obtained the best results by only training the feed forward network. For the backbone and transformer we used the pre-trained weights trained on the coco data set. DETR outperformed the previous model, Faster RCNN with inception v2 backbone, that was trained on this data set. The previous model achieved an AP of 0.6012 (AP@[IoU=0.5:0.95]). But to archive that, it had to use a second class "background" to prevent false positives. This second class was not used for the fine-tuning of DETR. One of the predictions from the test set can be seen in figure 7.

Average Precision	IoU=0.50:0.95	maxDets=100	0.619
Average Precision	IoU=0.50	maxDets=100	0.928
Average Recall	IoU=0.50:0.95	maxDets=10	0.708
Average Recall	IoU=0.50:0.95	maxDets=100	0.755

Table II
Fine-tuned DETR on the pigeon data set. Parameters: Learning rate: $5 * 10^{-5}$, epochs: 380, backbone: ResNet50, Learning rate backbone: 0

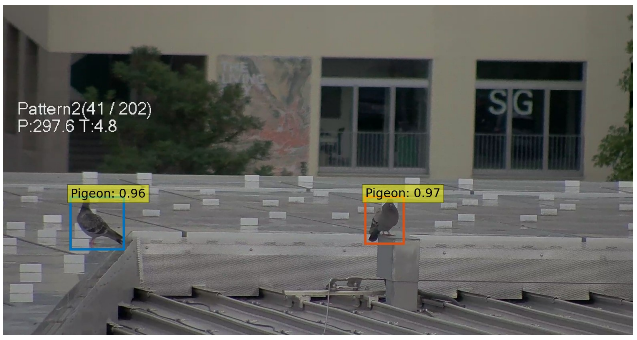


Figure 7. Prediction result for an image from the pigeons test set

C. Best Model Drones

For the drone data set we only have one model as it is the only one we successfully trained. As for the pigeon data set we obtained this result by fine-tuning the pre-trained DETR. The weights of the feed forward network were reset. Since the pre-trained weights were used, we could not implement the selected backbone (ResNet18) and had to use the "ResNet50" backbone. So we did not reach the goal of obtaining a model that evaluates in real-time on the Jetson. One of the predictions from the test set can be seen in figure 8.
This model could be improved by training for more epochs. Because of the tight time schedule we were only able to train for 100 epochs.

Average Precision	IoU=0.50:0.95	maxDets=100	0.422
Average Precision	IoU=0.50	maxDets=100	0.818
Average Recall	IoU=0.50:0.95	maxDets=10	0.530
Average Recall	IoU=0.50:0.95	maxDets=100	0.551

Table III
Fine-tuned DETR on the drone data set. Parameters: Learning rate: $5e^{-5}$, epochs: 100, backbone: ResNet50, Learning rate backbone: 0

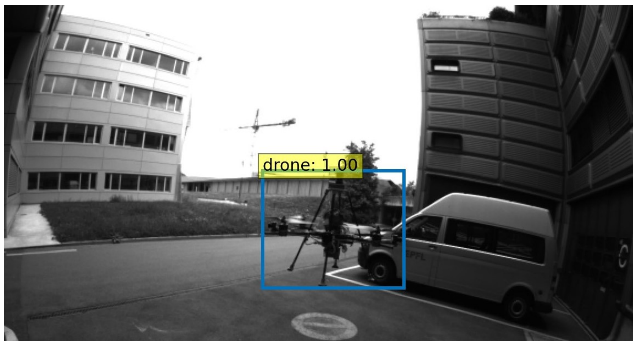


Figure 8. Prediction result for an image from the drones test set

IV. Summary

It was shown that it's possible to fine tune the DETR architecture to custom data sets in the case of drones and pigeons. Good average precision values were obtained. In the case of the pigeons data set DETR even outperformed a more complex model.
The goal of evaluating images in real time on the embedded hardware of the drones was not achieved, because we were unable to train DETR with a smaller backbone. The evaluation time for our model is 500ms on the Jetson.
It should be investigated why a full training with a different backbone did not succeed. Reasons could be a inappropriate learning rate or inappropriate loss coefficients.
Another approach to avoid training the transformer from scratch with a different backbone could be to use the pre-trained weights for the transformer, even thought they were obtained with a different backbone.

References

[1] Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. End-to-end object detection with transformers, 2020.

[2] Tsung-Yi Lin, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence Zitnick, and Piotr Dollár. Microsoft coco: Common objects in context, 2015.

[3] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks, 2016.

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

[5] Jetson TX2 Module, May 2017.

[6] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

[7] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks, 2017.

[8] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks, 2017.

[9] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks, 2019.

[10] Mingxing Tan and Quoc V. Le. Efficientnet: Rethinking model scaling for convolutional neural networks, 2020.