

# AWS Machine Learning Engineer Nanodegree

---

## Report

### Airplane Object Detection Project

#### Author:

Tomás Baidal

#### Abstract

This project has addressed the development and implementation of an object detection model focusing on identifying airplanes in images, using the MXNet framework. We will use a TensorFlow Object detection model as a benchmark model in order to compare our model performance and establish a threshold. In order to build our model, we will perform transfer learning leveraging a pre-trained ResNet-50 model to optimize the model's efficiency and effectiveness. Various optimization techniques and hyperparameter tuning have been applied to enhance the model's accuracy, assessed through the AP (Average Precision) and IoU metrics. Ultimately, the model has been integrated into a web application developed with AWS Lambda, AWS API Gateway and Python Dash framework, allowing users to upload images and receive detection results.

## 1. Definition

### Project Overview

This project encapsulates an experimental journey in deploying an AWS-based object detection model to identify airplanes in images. Using MXNet with ResNet50 and transfer learning, we developed a model and benchmarked it against TensorFlow's established object detection suite. The comparison wasn't solely about performance but served as a reflective exercise in understanding and iterating our approach. Beyond model creation, we integrated the solution into a user-friendly application using AWS Lambda, API Gateway, and Python Dash, allowing real-time airplane detection from user-uploaded images. This endeavor not only showcased end-to-end machine learning capabilities but also emphasized the practical applications of cloud microservices in addressing real-world challenges.

## Problem Statement

The primary objective of this project is to create a model that can identify airplanes within images, using advanced machine learning techniques. We embark on this initiative by leveraging the MXNet framework, deploying a pre-trained ResNet50 model, to accurately detect airplanes amidst diverse visual scenarios.

To benchmark the performance and set standards for our model's precision and reliability, we utilize a TensorFlow Object Detection model. This model serves as a comparative measure, allowing us to gauge the efficiency and accuracy of our MXNet-based solution.


The overarching challenge lies in adapting and optimizing the ResNet50 model to proficiently analyze the visual data, ensuring that it can discern projectiles with high reliability and consistency. This task extends beyond mere image classification; it's an object detection problem. Our model is tasked not only with categorizing the images but also with precisely localizing the airplanes within them, providing a comprehensive understanding of the scene and comparing it with the chosen benchmark.

## Metrics

The metrics used to evaluate and compare the performance of our models will be IoU (Intersection over Union) and mAP (Mean Average Precision).

### - IoU (Intersection over Union):

IoU is a metric used to measure the overlap between two bounding boxes. It's commonly used in object detection to compare the predicted bounding box of an object to the ground truth bounding box. The IoU is calculated by dividing the area of overlap between the two bounding boxes by the area of their union:

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


[Source](#)

The resulting value ranges from 0 (no overlap) to 1 (perfect overlap). A higher IoU indicates a more accurate detection.

In this project, we will measure the IoU after the inference of our validation dataset to see which model performs better.

#### - **mAP (Mean Average Precision)**

mAP is a metric used to assess the performance of object detection models, especially when dealing with a single object class, as in our case with airplanes. It averages the precision of the model over different levels of recall. In this context, precision quantifies how many of the detected airplanes are true positives, while recall captures how many of the actual airplanes in the images were correctly detected by the model.

In our project, we employed the Average Precision (AP) metric to assess the performance of our object detection models. This metric provides an average of precision across different recall levels and various Intersections over Union (IoU) thresholds. Through a systematic process, which includes sorting detections by confidence and computing precision and recall, we calculated AP values for both our TensorFlow and MXNet models

## **2. Analysis**

### **Datasets and Sources**

We will be working with extensive datasets derived from [Google's Open Images Dataset V7](#), organized into training and validation folders and dataframes. Data can be downloaded using a [simple python script in a local environment](#) using an open source tool called [fiftyone](#) and can not be downloaded in a Sagemaker environment as fiftyone is still not supported in AWS yet. Instructions to download the data are provided and data can be downloaded easily.

The datasets, in our case, consist of 900 images as a training set and 250 images for validation. We will split our training set into a training and test set, leaving the validation set to validate the model that we will build and inference the benchmarking model.

## Columns Overview:

Our datasets comprise several columns providing varied information about each image, such as ImageID, LabelName, bounding box coordinates (XMin, XMax, YMin, YMax), and other columns related to the state and position of the object in the image. We will only use the mentioned columns.

	ImageID	Source	LabelName	Confidence	XMin	XMax	YMin	YMax
0	000002b66c9c498e	xclick	/m/01g317	1	0.012500	0.195312	0.148438	0.587500
1	000002b66c9c498e	xclick	/m/01g317	1	0.025000	0.276563	0.714063	0.948438
2	000002b66c9c498e	xclick	/m/01g317	1	0.151562	0.310937	0.198437	0.590625
3	000002b66c9c498e	xclick	/m/01g317	1	0.256250	0.429688	0.651563	0.925000
4	000002b66c9c498e	xclick	/m/01g317	1	0.257812	0.346875	0.235938	0.385938

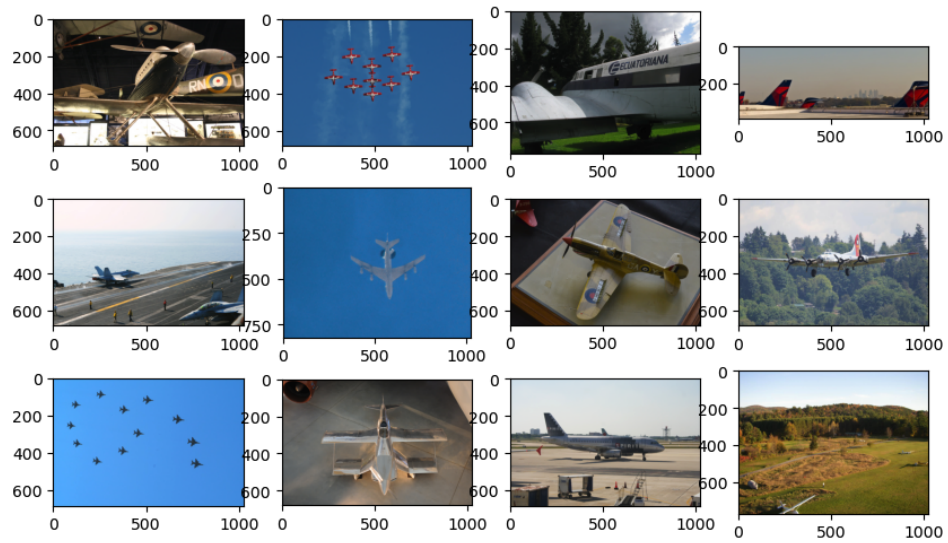
## Exploratory Visualization

We have performed several visualizations of our data, including images with one airplane and images with several:

```
# Random image from trainImages  
visualize_random_image("unzipped/trainImages/train/data/*.jpg")
```

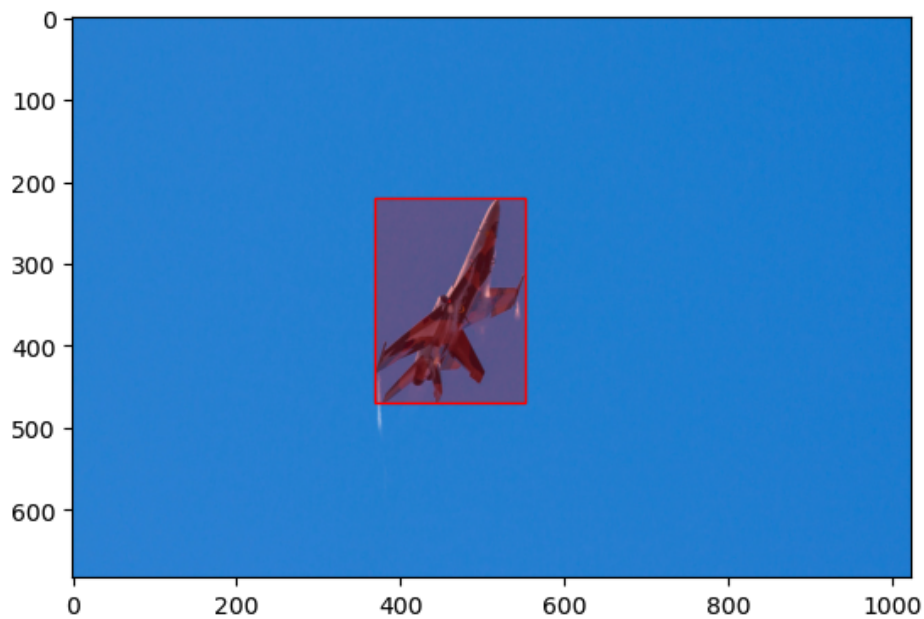


```
visualize_many(from_num=40, to_num=60, datasetpath="unzipped/trainImages/train/data/*.jpg")
```

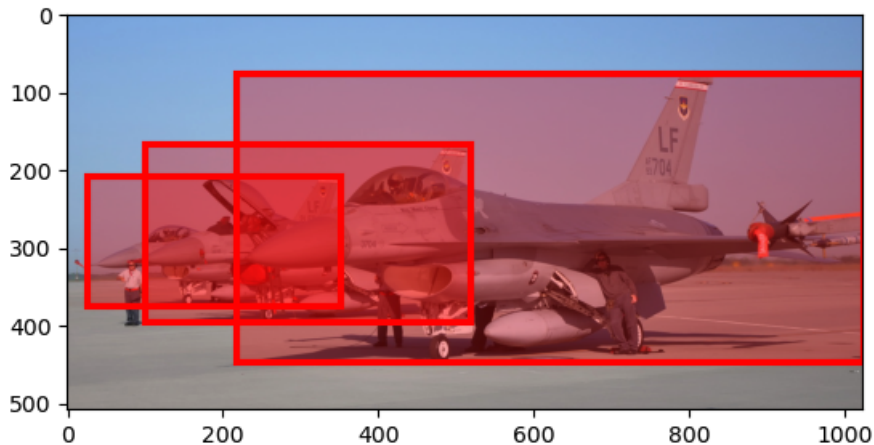


We have also visualize the bounding boxes, as it will be the ground truth of our project:

```
visualize_single_bbox(image_path, xmin, xmax, ymin, ymax)
```



```
# From Validation folder
visualize_bb_with_data("unzipped/validationImages/validation/data/", df=
```



## Algorithms and Techniques

### Data Augmentation:

Data augmentation is a technique used to artificially expand the size of a training dataset by applying various transformations to the original images. This process not only increases the volume of data but also introduces diversity, which can help improve the model's generalization capability.

In our project, we applied data augmentation by flipping the images along the Y-axis. This specific transformation effectively doubled our dataset's size, ensuring a broader representation and potentially enhancing the robustness of our model against varied inputs. This technique is explained visually in a notebook *04.1-data-augmentation-playground.ipynb*

### RecordIO files Creation:

RecordIO is a highly efficient binary data format often used within the MXNet framework. Designed for performance, the RecordIO format ensures streamlined and fast data I/O operations, making it an ideal choice for large-scale machine learning tasks. When training deep learning models like ResNet50, the speed at which data is fed into the model becomes a crucial factor in the overall training time. We will convert our training and test data into .rec format, to optimize this data ingestion process, enabling faster iterations and more efficient utilization of computational resources.

## **Transfer Learning:**

[Transfer learning](#) is a powerful technique where a neural network, previously trained on a large dataset (often a general-purpose dataset like ImageNet), is fine-tuned for a new, often related, task. The rationale behind this approach is the knowledge gained while learning one task can aid performance on a related task. For this project, transfer learning is indispensable. Training object detection models from scratch demands significant computational resources and vast amounts of data. By leveraging the patterns already learned by a model on a comprehensive dataset, we can adapt it to our specific airplane detection task with our limited dataset, ensuring robustness.

## **ResNet-50:**

ResNet-50 is part of the ResNet family, a deep neural network architecture renowned for its depth and its ability to address the vanishing gradient problem using [residual connections](#). These connections enable the training of much deeper networks by allowing gradients to bypass layers, essentially creating shortcut connections. Given its depth, ResNet-50 can capture intricate features in images, which is crucial for precise object detection. For our project, ResNet-50 was chosen as the backbone due to its proven effectiveness in image classification and detection tasks. We will apply the Transfer Learning technique on this deep neural network.

## **Single Shot MultiBox Detector (SSD):**

The [SSD algorithm](#) is an object detection algorithm that excels in predicting multiple bounding boxes and class scores for each box during a single forward pass of the network. Its architecture is optimized for efficiency, ensuring it functions as a rapid detector without sacrificing significant accuracy. SSD's unique combination of speed and precision makes it a fitting choice for these needs. Importantly, the SSD technique is the backbone of our TensorFlow benchmarking model.

## **Benchmark**

The benchmark model used to compare our MXNet ResNet-50 model will be an Object-detection model of TensorFlow. The reason is because it is a well-known model used for object detection and it was trained using the COCO2017 dataset, a large-scale dataset designed for various computer vision tasks.

The chosen benchmark model boasts a COCO mAP (Mean Average Precision) of 38.3, showcasing its robustness and capability in the object detection arena. One of the detectable object classes in this model is "airplane," which aligns with our project's primary focus.

Given the model's pre-training on such a comprehensive dataset, it serves as a formidable baseline for performance evaluation. To establish a threshold and assess the efficiency of our MXNet ResNet-50 model, which employs transfer learning, we subjected the TensorFlow benchmark model to our validation dataset, comprising 250 images. By comparing the results of the benchmark model with our custom-built MXNet model, we aimed to gauge the relative strengths and areas of improvement in our approach.

In essence, the TensorFlow benchmark model provided a reference point, enabling us to measure and reflect upon the efficacy of our tailored solution in the context of airplane detection.

### **3. Methodology**

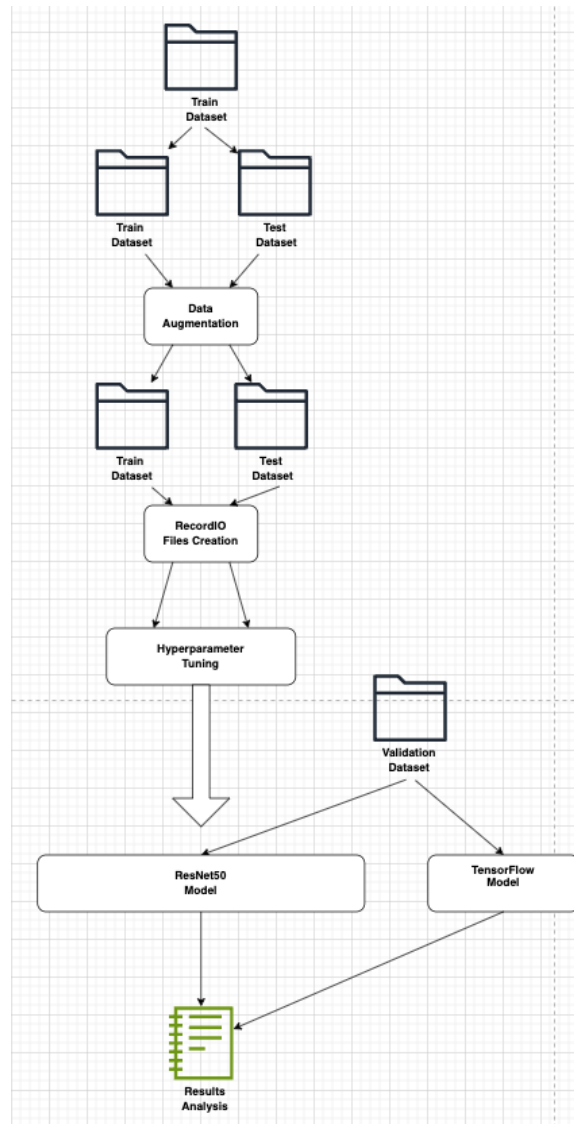
#### **Data Preprocessing**

The methodology carried out is divided into different notebooks. With this, we have modularized the project to make it easier to review and understand the entire process. In the *utils* folder are located the scripts that contain the functions we have used throughout the project, mostly in the parts where we have manipulated and filtered our images and datasets.

Additionally, this modular approach ensures that each step of the project is isolated, making it straightforward to identify, troubleshoot, and modify specific parts of the project as needed. This structure also promotes better code organization and management, fostering a more efficient and collaborative development environment.

The following flowchart reflects the pre-processing path of the project from data collection to analysis of the model results:





Project Notebooks:

## 01. EDA and Data Visualization:

In this first notebook we have loaded our datasets and filtered our class “airplane”. We have also counted the number of images and rows per dataset, checking that both are aligned. To finalize we have performed some visualizations in order to see the images that we will use to build our model and we have saved our filtered csv for the next step.

## 02. Data Preprocessing

In this second notebook, we have splitted our train set into a train and test set (80-20). The results were 773 images for the train set and 260 images for our test set. We have also moved and copied the test images to a separate folder and saved our new train and test csv's for the next process.

### 03. Feature Modelling

In this notebook we have prepared our train and test set in order to create our RecordIO files that we will use in order to train our MXNET model. Columns have been modified to the following label format:

***header\_cols label\_width className XMin YMin XMax YMax ImageID***

As described in the [documentation](#), this is the proper format in order to create the RecordIO files that we will use in our hyperparameter tuning.

### 04. Data augmentation

Before the process of creating the RecordIO files, we performed a [data augmentation](#) task, consisting in creating a copy of our images, a copy reflection of the original ones flipping it across the Y Axis. This allowed us to multiply by two the number of images. We have also did the same with the rows in the dataset. The result of the data augmentation has been:

Dataset	Condition	Images in Folder	Rows in Dataset
TRAIN	Before Augment	773	1352
TRAIN	After Augment	1546	2704
TEST	Before Augment	260	338
TEST	After Augment	520	676

As described, we will always have more rows than images in our sets, as some images have more than one airplane and each row points to a detection in an image and not to an image itself.

#### 0.4.1 Data Augmentation Demo

To better understand, we have created a notebook in order to explain how the data augmentation technique works and how we implemented the technique.

### 05. Benchmarking

Next, we have taken an intermediate step in order to perform our inference in our chosen benchmarking model by deploying the TensorFlow model creating an endpoint and passing the 250 images that compose our validation set. We have saved the predictions as we will use them to compare and extract results.

The code to deploy and infer the benchmark model has been used following [this example that TensorFlow provides](#), so the process has been straightforward and easy to implement as we did not have to fine tune the model as the model was trained to detect airplanes.

## **06. MXNet RecordIO format Engineering**

Finally, we have performed our RecordIO files generation, [using a script provided by Apache](#) that easily creates our train and test RecordIO files that our training job will use. Process was also quick and took only 2 minutes in order to create the new files that we will use in our hyperparameter training job.

## **07. Hyperparameter Tuning - Model Deployment - Inference**

This is one of the most important notebooks of our project as in this project we will create, deploy and inference our new model. The [hyperparameters](#) used have been:

- **Learning Rate (0.001 to 0.1):**

Chosen to explore a common range for learning rates, ensuring the model can learn at varying speeds. The lower bound allows the model to learn slowly, possibly achieving a more refined and accurate model, while the upper bound allows for faster learning which could be beneficial in cases where the optimal solution is easy to find.

- **Mini Batch Size (8, 16):**

These values were selected to explore the trade-off between the computational efficiency and model's ability to generalize. A smaller batch size may offer more robust convergence properties and generalization, while a larger batch size allows for computational efficiencies due to optimized matrix operations and parallelization capabilities.

- **Optimizer (SGD, Adam):**

These are two widely used optimization algorithms in training deep learning models. SGD is known for its robustness and efficiency, and Adam is recognized for its effective adaptive learning rate properties, allowing the exploration of the benefits of both momentum and adaptive learning rates during the tuning process.

- **Epochs (50, 60, 70, 80, 95):**

These values were chosen to give the model ample opportunity to learn from the data iteratively and converge while preventing potential overfitting. Different epoch values allow us to observe the model performance at various points in the learning process, helping in understanding the learning dynamics and identifying the most suitable training duration for the model.

- **Max Jobs (6):**

This was selected to ensure a broad exploration of the hyperparameter space, allowing multiple training jobs with different hyperparameter combinations. It allows for a balance between gaining insights and managing computational resources and costs effectively.

- **Max Parallel Jobs (1):**

Set to one to execute training jobs sequentially, ensuring that the computational resources are not overwhelmed by running multiple training jobs simultaneously. This setting aids in maintaining system stability and avoiding potential resource contention, leading to more reliable training job outcomes.

- **Early Stopping Rule (60 Steps):**

Implemented to monitor the model's training process and stop the training if the loss does not decrease for 60 consecutive evaluations, ensuring a minimum of 2 epochs are run. This helps in preventing unnecessary computations and possible overfitting when the model is not improving, contributing to more efficient resource usage and potentially better model generalization.

### **Cost Considerations in Hyperparameter Selection:**

The hyperparameter choices for this project were carefully made, acknowledging both optimal model performance and the constraints of computational costs. Using the **ml.p3.2xlarge** instance, while powerful, **led to higher expenses than initially anticipated.**

After the tuning job, we got the following results:

	learning_rate	mini_batch_size	optimizer	TrainingJobName	TrainingJobStatus	FinalObjectiveValue	TrainingStartTime	TrainingEndTime
0	0.100000	8.0	sgd	object-detection-231007-1859-006-6e7f464d	Completed	0.000527	2023-10-08 04:51:02+00:00	2023-10-08 06:51:55+00:00
1	0.006489	8.0	sgd	object-detection-231007-1859-005-8923062a	Completed	0.400602	2023-10-08 02:42:07+00:00	2023-10-08 04:48:31+00:00
2	0.015592	16.0	sgd	object-detection-231007-1859-004-ce692836	Completed	0.404084	2023-10-08 00:51:33+00:00	2023-10-08 02:33:11+00:00
3	0.003661	16.0	sgd	object-detection-231007-1859-003-b005d6f5	Completed	0.397920	2023-10-07 23:01:14+00:00	2023-10-08 00:43:58+00:00
4	0.024009	8.0	adam	object-detection-231007-1859-002-eb707769	Completed	0.045820	2023-10-07 20:50:58+00:00	2023-10-07 22:51:55+00:00
5	0.069350	16.0	adam	object-detection-231007-1859-001-6adfa61c	Completed	0.027924	2023-10-07 19:00:44+00:00	2023-10-07 20:39:52+00:00

Having as a best training job: *object-detection-231007-1859-004-ce692836* with the following metrics:

	0	1	2	3	4
MetricName	train:progress	train:smooth_l1	train:cross_entropy	validation:mAP	ObjectiveMetric
Value	100.0	0.259656	0.523478	0.404084	0.404084

and parameters:

	HyperParameter	Value
0	_tuning_objective_metric	validation:mAP
1	base_network	resnet-50
2	epochs	100
3	image_shape	512
4	learning_rate	0.015591662535987936
5	lr_scheduler_factor	0.1
6	lr_scheduler_step	50,70,80,90,95
7	mini_batch_size	16
8	momentum	0.9
9	nms_threshold	0.45
10	num_classes	1
11	num_training_samples	1546
12	optimizer	sgd
13	use_pretrained_model	1
14	weight_decay	0.0005

Following this, we have deployed the model using boto3 in order to inference in our new model:

```
runtime = boto3.client(service_name = "runtime.sagemaker")

# Model artifact location in S3
best_model = f"s3://airplane-object-detection-project/model/model_output/object-detection-231007-1859-004-ce692836/output/
best_model

's3://airplane-object-detection-project/model/model_output/object-detection-231007-1859-004-ce692836/output/model.ta
r.gz'

model = sagemaker.model.Model(
    image_uri = training_image,
    model_data = best_model,
    role = role)

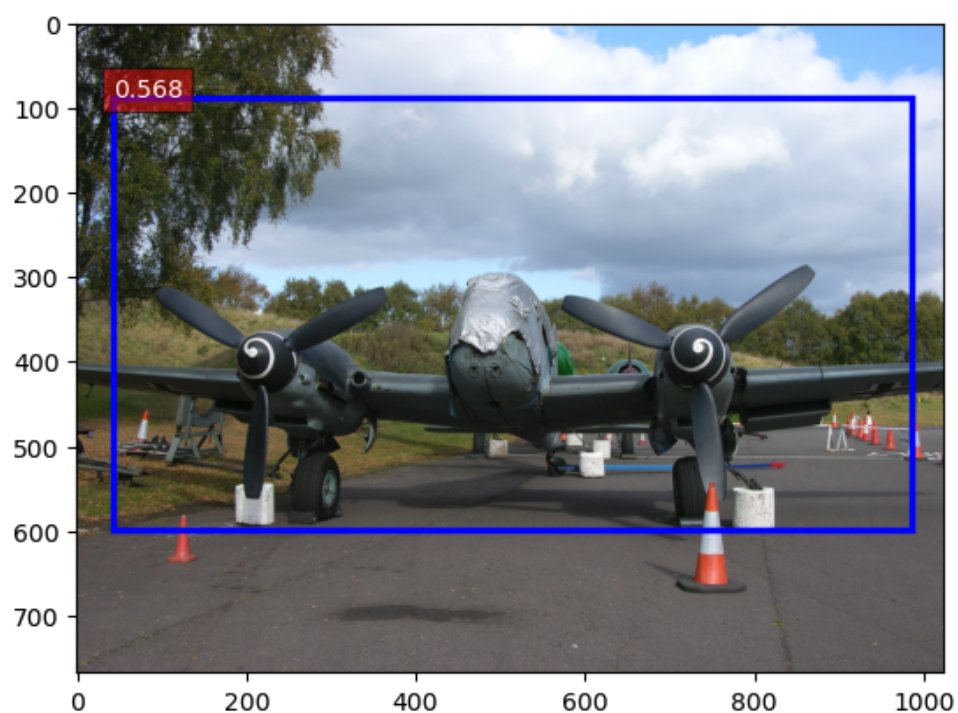
sagemaker.config INFO - Not applying SDK defaults from location: /etc/xdg/sagemaker/config.yaml
sagemaker.config INFO - Not applying SDK defaults from location: /root/.config/sagemaker/config.yaml

endpoint_name = "airplane-detection-endpoint"

deploy_model = model.deploy(
    initial_instance_count = 1,
    instance_type = "ml.m4.xlarge",
    endpoint_name = endpoint_name)

sagemaker.config INFO - Not applying SDK defaults from location: /etc/xdg/sagemaker/config.yaml
sagemaker.config INFO - Not applying SDK defaults from location: /root/.config/sagemaker/config.yaml
-----!
```

To finalize this notebook we have tested the new model with some images:



And then we perform the inference on the validation set and save the results for the validation analysis.

## **08. Validation Results Analysis**

In this notebook we brought the results on the inference on the chosen TensorFlow benchmark model, the MXNET ResNet50 model we created using transfer learning and the ground truth validation set in order to analyze, explore and investigate the results and choose a model in order to deploy it on the cloud:

The process to analyze the results of the inferences using the data validation has been explained in the following steps:

- a. Loading results of the inferences on MXNET Model, TensorFlow model and the validation dataset.
- b. Merging data and calculate metrics:
  - i. IoU of TensorFlow model compared with ground truth dataset
  - ii. IoU of MXNet model compared with ground truth dataset
  - iii. Plotting basic statistics: mean, std, min, max and percentiles.
  - iv. Distributions plots and distributions analysis
  - v. Visualizations
  - vi. AP Calculation, logic explanation and results interpretation
  - vii. Precision-Recall Curve visualization and explanation
  - viii. Final statement and reason of model chosen model to deploy

## **09. AWS Lambda and AWS API Gateway setup**

Here we explained the process to deploy the model and use an API to inference on it and display it on the Python Dash app:

- Firstly we will create a Lambda Function that will interact with the model, passing the image and returning the prediction.
- Secondly we will create our API Post request resource that will bring the request to the lambda and give the prediction to the user as a response.

### **09.1 Test API Endpoint Model**

We have tested our new API using a simple notebook and making POST request in order to get predictions from our new deployed model.

## 10. Dash App Development

Here we have created a simple dash app that will serve as an interface between a user and our model. Users will be able to upload or drag and drop an image, and receive the image with the bounding box predicted by the model.

## 4. Results

The results of the model inferences, the metric extraction process and the study of the metrics have been computed in the notebook **08-validation-results-analysis**.

After the results loading into Pandas dataframes we have visualized the content of the data in order to plan how to analyze and get results:

MXNet Inference Results overview:

```
MX = transform_dataset_mx(mx, "mx")
MX.head()
```

	image_id_mx	xmin_mx	ymin_mx	xmax_mx	ymax_mx	confidences_mx
0	13c7df326952233f	0.155785	0.444461	1.000000	0.672971	0.638986
1	13c7df326952233f	0.153975	0.195009	0.951689	0.733860	0.330336
2	376c88a2570886f3	0.146672	0.140796	0.972069	0.519564	0.519519
3	483ac6e281e1ece5	0.007536	0.033724	0.987060	0.974369	0.856118
4	35e4af0afb6303a	0.188064	0.319048	0.801040	0.543995	0.692767

TensorFlow Inference Results overview:

```
TF.head()
```

	image_id_tf	xmin_tf	ymin_tf	xmax_tf	ymax_tf	confidences_tf
0	13c7df326952233f	0.195158	0.370841	0.961156	0.659080	0.825679
1	376c88a2570886f3	0.340363	0.191415	0.999032	0.446537	0.335392
2	483ac6e281e1ece5	0.006360	0.002514	1.000000	0.992113	0.654442
3	35e4af0afb6303a	0.243866	0.331120	0.793745	0.530109	0.678474
4	01b95ffb7ee0752c	0.000511	0.048265	0.988637	0.980944	0.692324



Validation dataset overview:

```
val.head()
```

	ImageID	XMin	XMax	YMin	YMax
0	0001eeaf4aed83f9	0.022673	0.964201	0.071038	0.800546
1	0009bad4d8539bb4	0.294551	0.705449	0.340708	0.515487
2	0019e544c79847f5	0.000000	0.349558	0.106195	0.396018
3	0019e544c79847f5	0.538348	0.874631	0.688053	0.909292
4	007384da2ed0464f	0.000000	1.000000	0.372917	0.768750

As we see, all datasets will be composed by:

- ImageID column
- XMin, XMax, YMin, YMax
- Confidences in the case of the results of the inferences

With this we have studied first the confidences of the inferences

## - Confidences:

### TensorFlow - Benchmarking

```
: tf_mean_confidence = TF["confidences_tf"].mean()
tf_min_confidence = TF["confidences_tf"].min()
tf_max_confidence = TF["confidences_tf"].max()

print(f"Confidence Mean: {tf_mean_confidence}")
print(f"Confidence Min: {tf_min_confidence}")
print(f"Confidence Max: {tf_max_confidence}")

Confidence Mean: 0.4915908181119632
Confidence Min: 0.200010106
Confidence Max: 0.938924074
```

#### TensorFlow Model Confidence Analysis:

- Mean Confidence: 0.4915
- Minimum Confidence: 0.2000
- Maximum Confidence: 0.9389

This analysis indicates that, on average, the TensorFlow model has a moderate confidence in its predictions, with a peak value nearing 0.94. The minimum confidence

threshold observed is 0.2000, which suggests there might be some predictions where the model is less sure.

It's worth noting that every image in the validation set has received a prediction from the model, as we are considering all confidences regardless of their value. This provides a comprehensive view of the model's performance across different confidence levels, and helps in understanding how the model behaves even in situations where it might not be very confident in its predictions.

## MXNet

```
mx_mean_confidence = MX["confidences_mx"].mean()  
mx_min_confidence = MX["confidences_mx"].min()  
mx_max_confidence = MX["confidences_mx"].max()  
  
print(f"Confidence Mean: {mx_mean_confidence}")  
print(f"Confidence Min: {mx_min_confidence}")  
print(f"Confidence Max: {mx_max_confidence}")
```

```
Confidence Mean: 0.394905679267178  
Confidence Min: 0.20002177357673645  
Confidence Max: 0.8627170920372009
```

**For the MXNet model, we observed the following confidence metrics:**

- **Mean Confidence: 0.3949**
- **Minimum Confidence: 0.2000**
- **Maximum Confidence: 0.8627**

MXNet model confidence statistics achieved a mean confidence of 0.3949. The model's predictions varied from a minimum confidence of 0.2000 to a maximum of 0.8627, highlighting its range of certainty across different detections.

The TensorFlow model has a slightly higher average certainty in its predictions compared to the MXNet model. The confidence scores ranged from a consistent minimum of 0.2000 to a notably higher maximum of 0.9389.

## - Intersection over Union (IoU):

After the confidences, we have calculated the metric *Intersection over Union* (IoU) of each prediction of the results of the models inferences with the validation set.

The following figures were obtained:

#### IoU Statistics TensorFlow inference:

	count	mean	std	min	25%	50%	75%	max
iou_tf	3471.0	0.622355	0.250236	0.082888	0.433179	0.584662	0.875058	0.997676

#### TensorFlow Model (IoU):

The TensorFlow model, on average, yields a 62.24% overlap with the ground truth boxes, indicating a commendable accuracy. The spread of its IoU values is somewhat broad, with the worst overlap being about 8.29% and the best nearing perfection at 99.77%. This variability suggests that while the model often predicts accurately, there are instances where it might falter significantly.

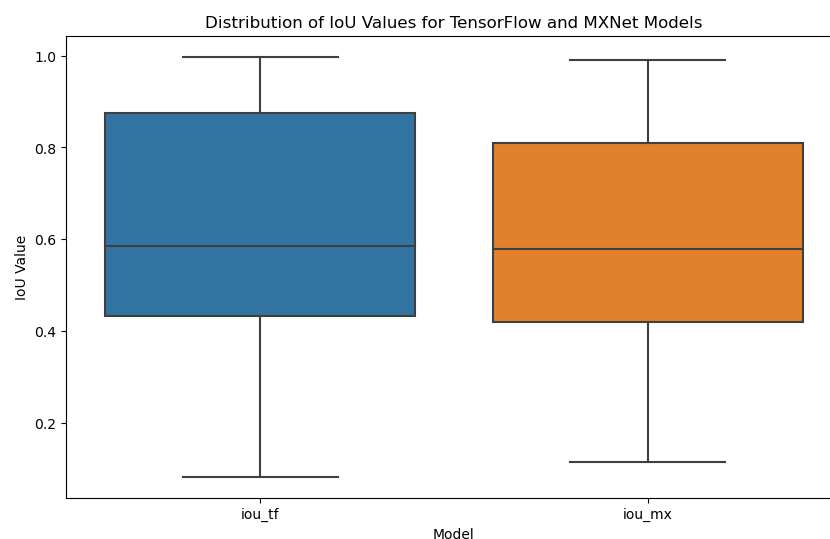
#### IoU Statistics MXNet inference:

	count	mean	std	min	25%	50%	75%	max
iou_mx	3473.0	0.607205	0.22704	0.115342	0.418799	0.577802	0.810452	0.990753

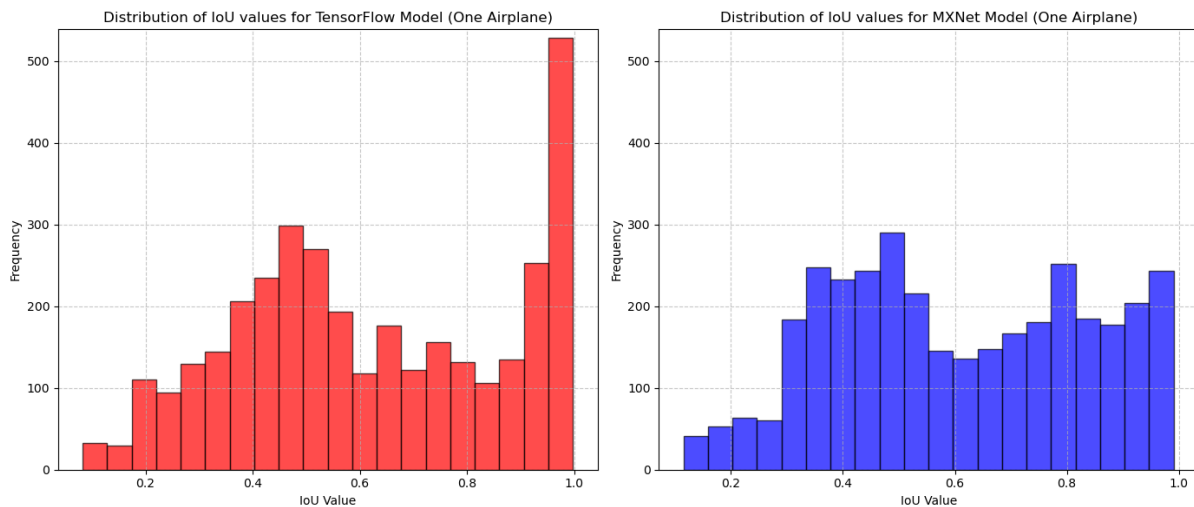
#### MXNet Model (IoU):

The MXNet model averages a 60.72% overlap with the actual bounding boxes, slightly trailing the TensorFlow model. Its predictions are more consistent, given the narrower spread of IoU values. While its minimum overlap is better than TensorFlow's at 11.53%, its peak performance, at 99.08%, is slightly below that of TensorFlow. This consistency suggests that the MXNet model provides a more stable prediction quality across different images.

We can see the statistics displayed on boxplots:



The next step has been to analyze the distribution of the IoU values of both inferences:



#### **TensorFlow Model (Red Histogram):**

- The IoU values for the TensorFlow model are most densely clustered around the range of 0.85 to 0.95. This indicates a substantial overlap with the ground truth bounding boxes for many predictions. However, there are more instances where the IoU values drop below 0.85 compared to your previous results, pointing to some predictions that might be less accurate.

#### **MXNet Model (Blue Histogram):**

- The distribution of IoU values for the MXNet model shows a significant concentration in the range of 0.75 to 0.85. While this is commendable, it's slightly less consistent than the TensorFlow model. There are also instances where the IoU values are as low as 0.4, suggesting that for some images, the MXNet model's predictions significantly deviate from the ground truth.

The histograms and statistics continue to underscore the TensorFlow model's strength in predicting bounding boxes that are very close to the ground truth. The MXNet model, while exhibiting decent performance, trails behind the TensorFlow model in terms of average IoU.

Given the computational resources and constraints during the model development and training phase, the performance of our MXNet model remains praiseworthy.

As we continue to improve, there are several areas to focus on:

- **Further Hyperparameter Tuning:** Exploring more combinations or different hyperparameter values might enhance the model's performance.
- **Model Architecture Adjustments:** Tweaking the current architecture or experimenting with alternative ones could yield superior results.
- **Data Augmentation:** Incorporating a wider variety of data augmentation techniques might enhance the model's generalization capabilities.
- **Transfer Learning:** Utilizing pre-trained models or weights can offer our model a more advantageous starting position, possibly resulting in better performance.

The TensorFlow benchmark model might have benefited from more extensive resources or a more comprehensive training and fine-tuning process.

## - mAP (Mean Average Precision)

To evaluate the performance of our object detection models on the dataset, we employ the metric known as the Mean Average Precision (mAP). The mAP provides a single figure that represents the average over multiple *Intersection over Union* (IoU) thresholds. A higher mAP indicates better overall performance of the model in terms of precision and recall.

Steps involved to calculate mAP using Python:

- **Sorting Detections by Confidence:** For each IoU threshold, detections are sorted by their confidence scores.
- **Determine TP or FP:** For each detection, it's categorized as a true positive (TP) or false positive (FP) based on its IoU with the ground truth.
- **Calculate Precision and Recall:** As we traverse the sorted detections, precision and recall values are cumulatively computed.
- **Interpolating Precision:** This step ensures the precision curve is monotonically decreasing, by replacing each precision value with the maximum precision value to its right.
- **Compute Average Precision (AP):** AP is the area under the precision-recall curve.
- **Average over IoU thresholds:** The final mAP is obtained by averaging the AP values over several IoU thresholds (from 0.5 to 0.95).

The function can be found in *utils/model\_analysis.py*

```
# Calculate mAP for TensorFlow model and MXNet model
mAP_tf = calculate_mAP(merged_data, 'confidences_tf', 'iou_tf')
mAP_mx = calculate_mAP(merged_data, 'confidences_mx', 'iou_mx')

print(f"mAP for TensorFlow model: {mAP_tf:.4f}")
print(f"mAP for MXNet model: {mAP_mx:.4f}")

mAP for TensorFlow model: 0.4596
mAP for MXNet model: 0.4067
```

### Benchmark Model (TensorFlow):

With a mAP of 0.4596, the benchmark model demonstrates a decent performance in detecting planes in our images. This efficiency could stem from its well-optimized architecture and potentially more extensive training or fine-tuning.

### ResNet-50 Model (MXNet):

The MXNet model, with a mAP of 0.4067, still shows a commendable ability to detect planes, albeit it lags slightly behind the TensorFlow model in terms of precision and recall. We've optimized our MXNet model as best as possible given our computational resource constraints, aiming to glean insights from its training and performance against a benchmark.

### Performance:

The gap in performance may arise from various facets, such as the architecture differences, the scope of hyperparameter tuning, and the model training duration.

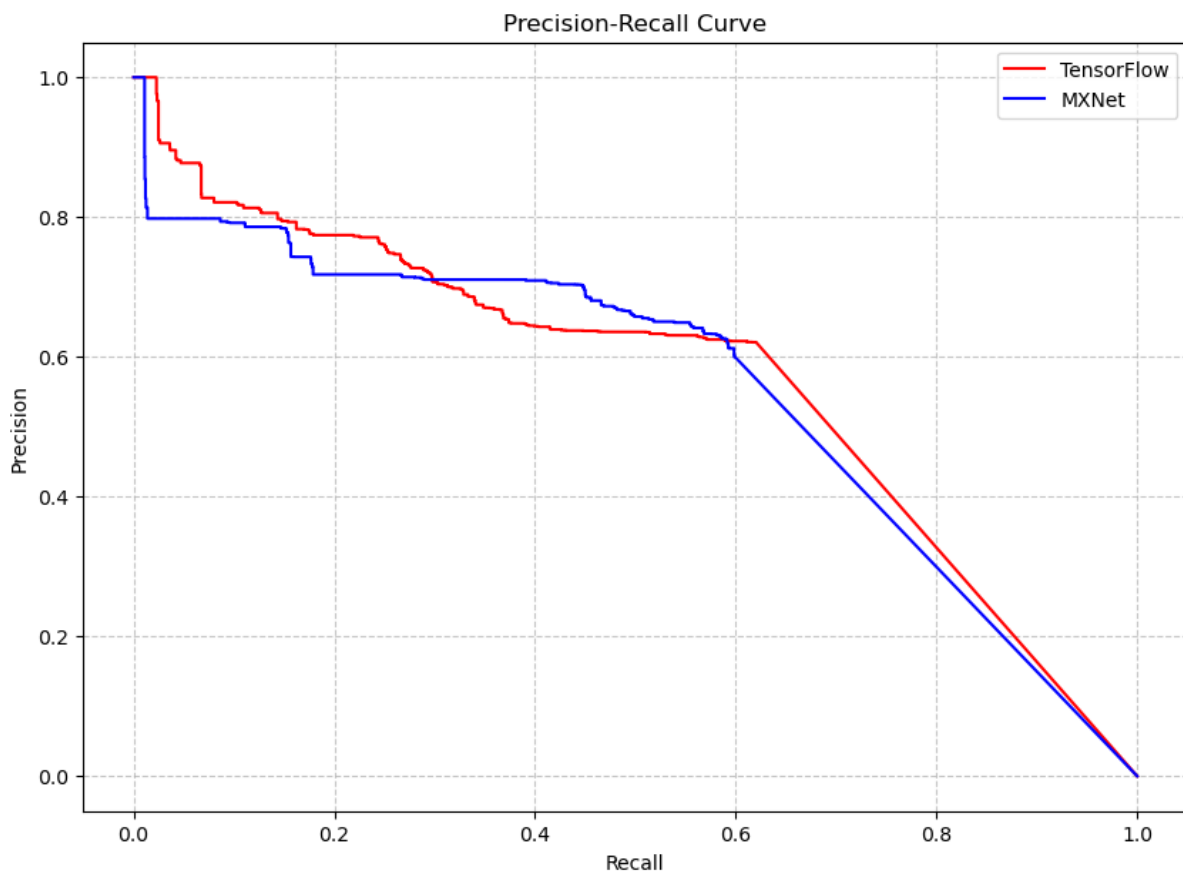
### Potential Improvements:

- **Data Augmentation:** Incorporating a more diverse range of data augmentation techniques can assist the model in better generalizing across different scenarios.
- **Model Architecture Enhancements:** Investigating deeper or alternative architectures might bring about improved outcomes.
- **Extended Hyperparameter Tuning:** With additional computational prowess, an exhaustive exploration in the hyperparameter realm might uplift the model's performance.
- **Regularization Techniques:** Deploying methods like dropout or batch normalization could bolster model robustness and generalization.

- **Transfer Learning:** Harnessing pre-trained models or weights from models that have been trained on related tasks can offer a significant advantage during training.

In summary, even though our MXNet model hasn't outperformed the benchmark, it provides invaluable insights into the nuances of model development, training, and performance tuning.

## - Precision-Recall Curve



The plot shows that the TensorFlow model has a higher precision than the MXNet model for all recall values. However, the difference in precision is larger for higher recall values. For example, for a recall of 0.8, the TensorFlow model's precision is 0.75, while the MXNet model's precision is 0.65.

This means that the TensorFlow model is better at detecting most of the airplanes in the images, but the MXNet model is better at detecting airplanes with a high degree of confidence.

In general, the TensorFlow model has a better performance than the MXNet model on this dataset. However, the MXNet model may be a good choice for applications where it is

important to detect airplanes with a high degree of confidence, even if this means sacrificing some precision.

## 5. Justification

For the deployment phase of our project, we've opted to proceed with our custom-built MXNet model, which was developed using transfer learning. There are several motivations behind this decision:

- **End-to-End Development:** Deploying the MXNet model exemplifies the complete lifecycle of our project, from initial conceptualization to final implementation. This encapsulates the essence of our project's objectives - to design, develop, and deploy a model on AWS built it to our specific requirements.
- **Educational Value:** This project was as much about the journey as the destination. Deploying our model provides invaluable learning experiences in model deployment and integration, essential skills in the broader machine learning domain.
- **Customization Potential:** While our MXNet model may currently trail the TensorFlow benchmark in terms of raw performance, its architecture and training process are entirely under our control. This offers significant flexibility for future optimizations and tweaks tailored to specific use cases or datasets.
- **Acknowledgement of Efforts:** The model symbolizes the collective efforts, challenges overcome, and knowledge gained throughout the project.
- **Pragmatic Consideration:** We recognize the superior performance of the TensorFlow benchmark model and, in a real-world production scenario, would lean towards its deployment for optimal results. However, for the scope and intent of this project, the MXNet model serves as a fitting representative of our endeavors.

In summary, the deployment of the MXNet model is a testament to our commitment to seeing a project through every stage, understanding the steps involved, and enjoying the learning and growth achieved.



## 6. References

### Evaluation Metrics:

- <https://jonathan-hui.medium.com/map-mean-average-precision-for-object-detection-45c121a31173>
- [https://www.researchgate.net/figure/mAP-Comparison-results-on-each-network-in-COCO-dataset\\_tbl6\\_348825339](https://www.researchgate.net/figure/mAP-Comparison-results-on-each-network-in-COCO-dataset_tbl6_348825339)
- <https://towardsdatascience.com/on-object-detection-metrics-with-worked-example-216f173ed31e>
- <https://towardsdatascience.com/introduction-to-object-detection-model-evaluation-3a789220a9bf>

### Object Detection:

- <https://medium.com/visionwizard/object-detection-4bf3edadf07f>
- <https://jonathan-hui.medium.com/object-detection-series-24d03a12f904>
- <https://pub.towardsai.net/maximizing-the-impact-of-data-augmentation-effective-techniques-and-best-practices-c4cad9cd16e4>

### Deployment:

- <https://austinlasseter.medium.com/deploying-a-dash-app-with-elastic-beanstalk-console-27a834ebe91d>
- <https://medium.com/@data.science.enthusiast/invoke-sagemaker-model-endpoint-with-aws-api-gateway-and-lambda-3d0c085dccb8>

### MXNet:

- <https://github.com/apache/mxnet/tree/master>
- <https://github.com/apache/mxnet/blob/master/tools/im2rec.py>