CSCI E-25

Object Detection

Steve Elston

Introduction to Object Detection

In other lessons we have primarily dealt with classification problems. We have worked with images with a single object which we classify into its category.

Object recognition is a more difficult task. The process allows us to deal with complex images containing multiple objects. There are two major sets in this process:

1. **Detect** the presence of each of multiple objects in an image. In other lessons, we have dealt with object classification, where there is a single object of interest in the image. Having an unknown number of objects in the image introduces considerable difficulty in the problem. Another difficulty is the trade-off between correctly detecting the objects of interest and falsely detecting spurious objects in the background.
2. Compute a minimum size **bounding box** around each object.
3. The object detected in each of the bounding box is **classified**.

Our goal is to give you an understanding of how modern object recognition algorithms work. Given the research momentum in this area it is entirely likely that there will be better performing algorithms by the time you read this.

Overview of Conventional Object Detection Algorithms

The object recognition problem has been around for quite some time. Historically, object recognition was performed using hand engineered features. By 2012 the **mean average precision** or **mAP** of object recognition had stagnated at around 40%. Only since 2013 there has been a surge of both interest and performance of object recognition with the introduction of deep learning methods. Since that time, the mAP achieved has continued to increase and is now routinely over 80%. These algorithms no longer require the cumbersome process of hand engineering features. Some key papers in the history of deep object recognition are:

1. In 2013 Erhan et. al. published Scalable Object Detection using Deep Neural Networks, which introduced the R-CNN algorithm the first widely accepted deep learning object detection algorithm. R-CNN demonstrated a significant improvement

in object recognition accuracy. However, this algorithm proved to be too slow for real-time video processing.
2. The Fast R-CNN algorithm was introduced by Girshick Fast R-CNN simplified the required computations but still struggled with real-time video.
3. Further improvements by Ren et. al. lead to the Faster R-CNN algorithm. However, the computational complexity of this algorithm was still rather high.
4. The Mask R-CNN algorithm was proposed by He, et. al. in 2018 The Mask R-CNN algorithm exhibits significantly improved object detection accuracy. This is particularly the case where there are large numbers of objects, such as flock of birds or a crowd of people. While not efficient enough for real-time video, Mask R-CNN should be considered if accuracy in complex scenes is required.

All of these algorithms share a similar architecture. This architecture is in the form of a pipeline with process steps computed sequentially. The key steps in these pipelines generally include:

1. **Convolutional Neural Network:** The CNN creates features which which are used to detect and then classify objects in the image.
2. **Candidate bounding boxes:** Candidate bounding boxes are generated. Multiple candidate bounding boxes cover each region.
3. **Filtering of bounding boxes:** The probability of an object being in each bounding box is computed and low probability boxes are filtered.
4. **Minimal bounding boxes:** The size of the bounding boxes must be adjusted to best fit the objects detected.
5. **Classification:** The objects in the bounding box are classified.

The result is that training and performing inference with these complex pipelines has significant computational complexity. Further, these pipelines can be difficult to train.

Object Detection with a Transformer Architecture

In this lesson you will explore a simple case of a farily state of the art object detection algorithm. This algorithm is based on a transformer architecture.

Transformer architectures are starting to dominat several areas of computer visions (CV). However, massive resources are required to effectively train these models. Here we will work with small model trained on a relativeluy small dataset. Our focus is on developing understanding of how vision transformers work, rather than optimizing performance.

The model we are working with uses 8 self-attention layers in the transformers. This model is based on the, now well know vision transformater model (ViT model), from An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale, Dosovitskiy, et.al., 2020 The model will be trained on the relatively small CIFAR 100 dataset,

containing samples of 100 object types. Both of these conditions will limit the accuacy of the trained model.

A number of Keras-based vision transformer models from Google Research are available. These models have been trained on massive datasets, and can be fine-tuned for spedicic problems. But be warned, working with these sophisticated models can be difficult.

The models in the Google GitHub repository include the model of Dosovitskiy, et.al., 2020. We used for this example. The details of pretraining the model in the repository are outlined in Appendix B of the paper, and include:

1. The semisupervised pretraning was performed for 1 million steps.
2. The semisupervised pretraining was performed using two very large dataset, the InageNet 21k dataset, size 1.3 TB, and JFT dataset of 300 million images.

In these exercises, we will use a variation of the ViT architecture suitable for opject detection. No pre-training is available for this model. Dispite training from scratch on a relaively small dataset the results achieved are reasonably good, but not state-of-the-art.

Theory of Object Detection

Bounding box parameterization

Object detection requires that we enclose each object with a bounding box. The parameters of the bounding box are computed using a regression model. The model is trained by minimizing the RMSE between the human marked boxes in the training data and the computed bounding boxes.

To apply the regression model, we need a parameterization for the bounding box. We do this with respect to **anchor boxes** which are **prior bounding box proposals**. The anchor boxes are transformed by both translation and scaling so that they best fit the object being detected.

Naively, we could simply parameterize the bounding box with a linear function:

$$b_x = t_x + c_x \tag{1}$$
$$b_y = t_y + c_y \tag{2}$$
$$b_w = p_w * t_w \tag{3}$$
$$b_h = p_h * t_h \tag{4}$$

Where,
b_x is the x coordinate of the center of the bounding box,
b_y is the y coordinate of the center of the bounding box,

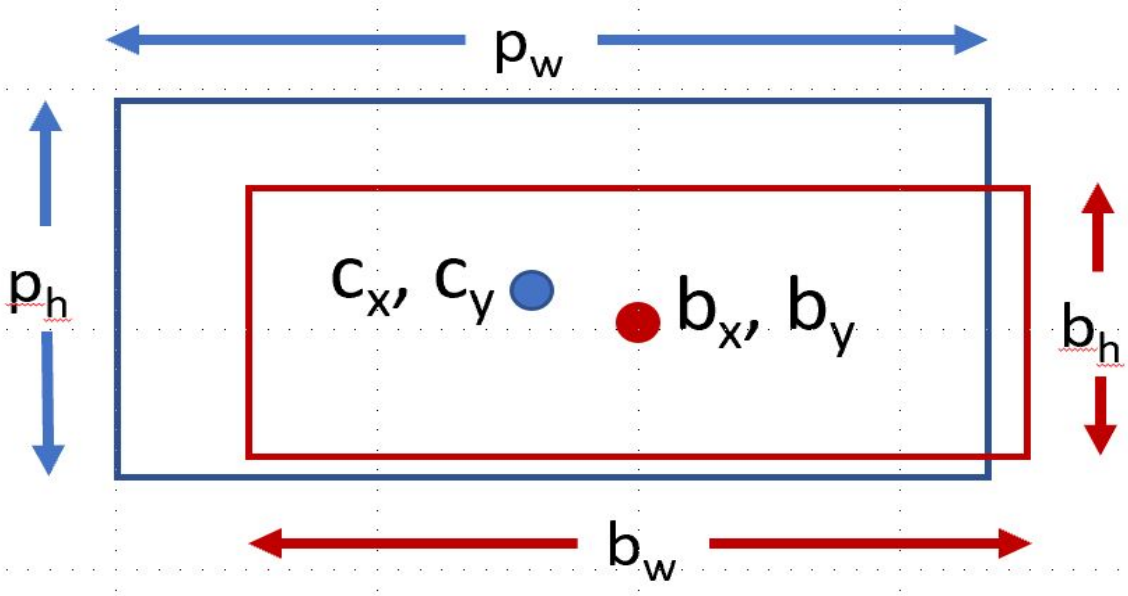
b_w is the width of the bounding box,
b_h is the height of the bounding box,
c_x is the prior of the x coordinate of the bounding box,
c_y is the prior of the y coordinate of the bounding box,
b_w is the prior of the width of the bounding box, and
b_h is the prior of the height of the bounding box.

There are 4 spatial parameters along with the probability that an object is in the box. For each bounding box the parameterization is:

$$b_x = \sigma(t_x) + c_x \tag{5}$$
$$b_y = \sigma(t_y) + c_y \tag{6}$$
$$b_w = p_w e^{t_w} \tag{7}$$
$$b_h = p_h e^{t_h} \tag{8}$$
$$p_0 = Pr(object) * IoU(b,\ object) = \sigma(t_0) \tag{9}$$

p_0 is the probability of an object in the box,

These relationships are illustrated in the figure below. The



Proposal bounding box in blue and optimized bounding box in red
The above relationships include priors, often known as proposals, for the parameters of the bounding box. These are the initial values of for the regression and are illustrated in the figure above in blue.

In the above figure the following notation is used to parameterize the best fit bounding box, which is shown in red: b_x is the x coordinate of the center of the bounding box,
b_y is the y coordinate of the center of the bounding box,
b_w is the width of the bounding box,

b_h is the height of the bounding box, and
p_0 is the probability of an object in the box.

Object detection by regression

Now that we have a parameterization of the object detection problem let's formulate the regression model. Using this parameterization, the best fit bounding box is computed using a regression model. Additionally, we need to classify the object in the bounding box. Our goal is to compute the smallest bounding box that encloses an object along with the probability there is an object in the box and the class of the object. By using a regression model to estimate all of the parameters

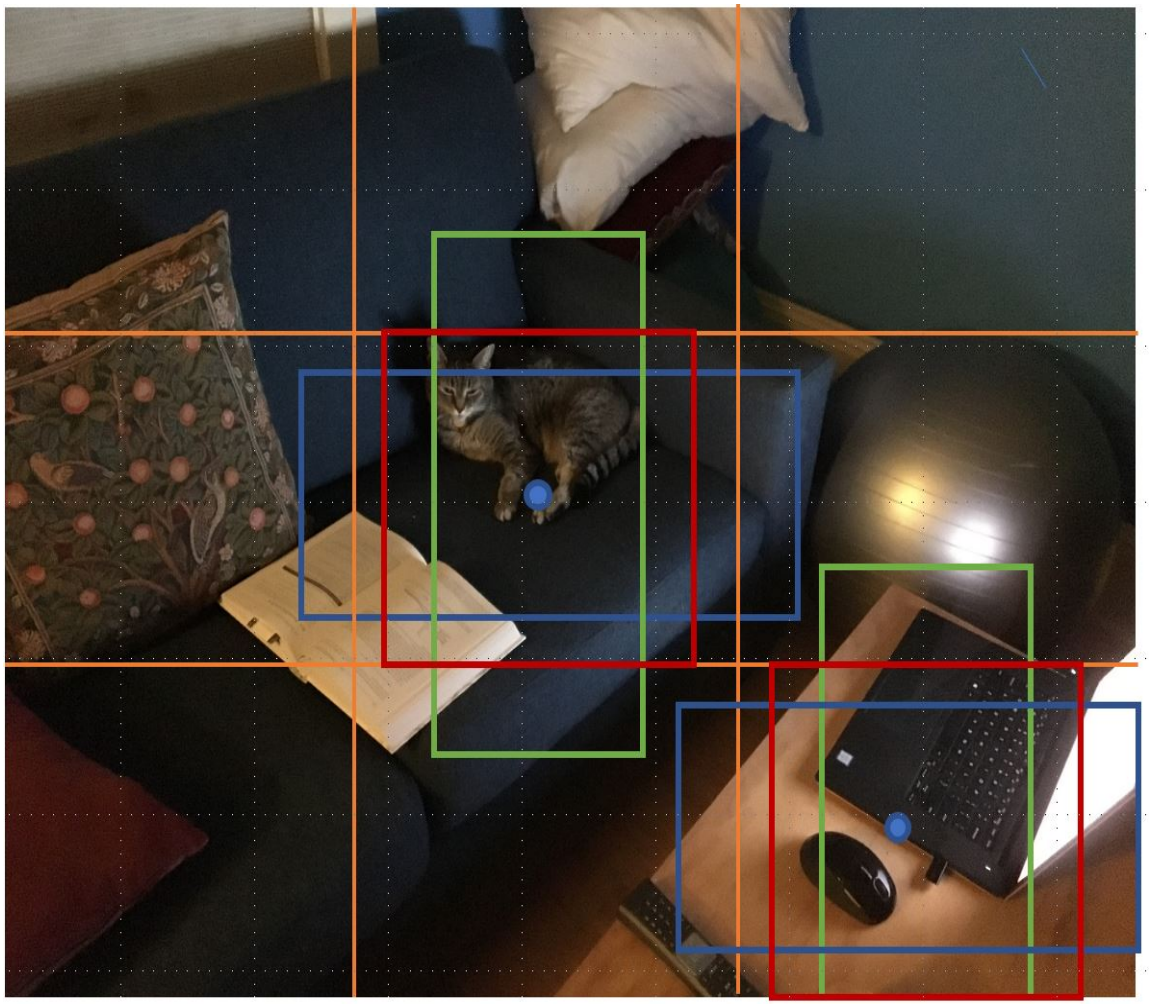
Training the regression model uses labeled or known cases. The known cases typically have human marked bounding boxes around known objects. The presence and class of the object are also labeled.

To illustrate the concepts we will start with just a single bounding box with three possible object classes. The vector of parameters to be estimated in this regression problem can be written as follows:

$$\hat{\boldsymbol{y}} = \begin{bmatrix} b_x \\ b_y \\ b_w \\ b_h \\ p_0 \\ c_1 \\ c_2 \\ c_3 \end{bmatrix}$$

The first 5 parameters in this vector have already been discussed. The values of $\{c_1, c_2, c_3\}$ are binary, and indicate which class the object falls in. Typically, the regression model is trained by minimizing the RMS error with the labeled values of the parameters.

Next, let's expand the simple single bounding box model to multiple bounding boxes. An example of multiple bounding box proposals sharing a single center, or **anchor**, are shown in the figure below. An object could have various shapes, resulting in a higher probability for one of the bounding boxes enclosing the object than the others.



Bounding boxes with common center or anchor point

There are two key points to note about the figure above.

1. The image is subdivided into a 3x3 grid. Bounding boxes are anchored on this grid.
2. For each anchor on the grid, there are three bounding boxes. The 3 bounding boxes for two anchors are shown.

In the case illustrated, there are 3 bounding box per-grid element for a total of 3x3x3 bounding box proposals. The YOLO algorithm typically uses a 9x9 grid with 5 bounding boxes per grid element, for a total f 9x9x5.

We can generalize the vector of model parameters discussed earlier for a single bounding box. For our example case, we concatenate 3x3x3 sets of parameters, one for each bounding box. An example of concatenating parameters for multiple bounding boxes is shown below.

$$\hat{\boldsymbol{y}} = \begin{bmatrix} b_x \\ b_y \\ b_w \\ b_h \\ p_0 \\ c_1 \\ c_2 \\ c_3 \\ \vdots \\ b_x \\ b_y \\ b_w \\ b_h \\ p_0 \\ c_1 \\ c_2 \\ c_3 \end{bmatrix}$$

As has already mentioned, the YOLO algorithm there are 9x9x5 bounding box proposals. If there are 80 classes of objects, plus a background class, then there are 9x9x5x81 model parameters.

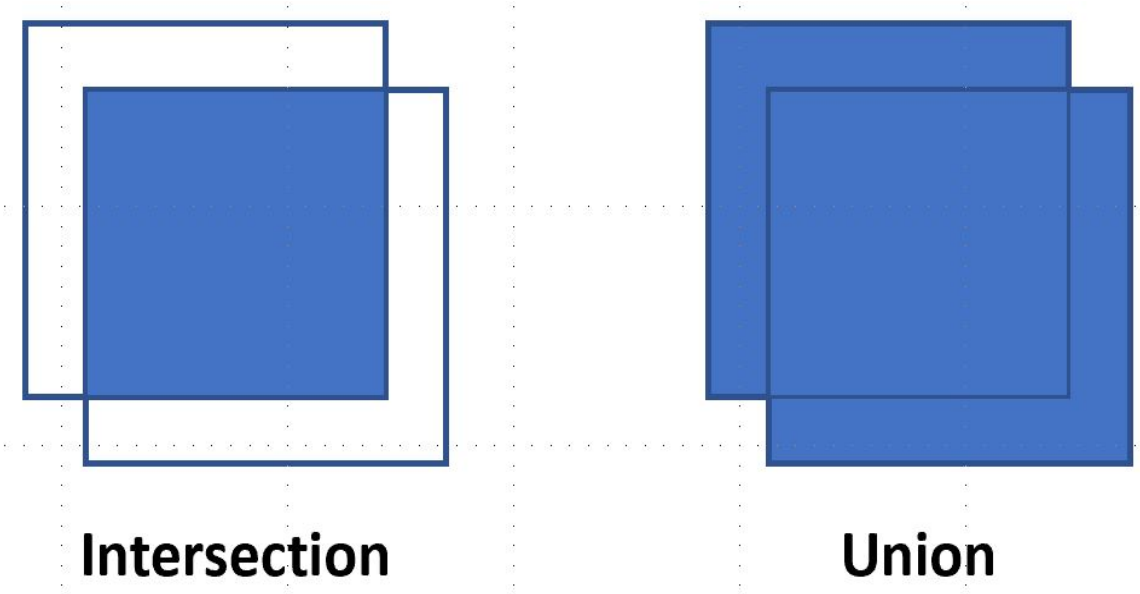
Filtering bounding boxes

The result of the regression model are a large number of bounding box proposals. We need an efficient algorithm

IOU

Object detection requires a metric to determine how well the computed bounding box fits the object. The metric we use is known as Intersection over Union or IoU. This metrics is used to compare human marked bounding boxes with the ones computed by the algorithm.

The concept of IoU is illustrated in the figure below:



Intersection and union of bounding boxes

Given the two bounding boxes the IoU is easily computed as:

$$IoU = \frac{Area\ of\ intersection}{Area\ of\ union}$$

Non-max Suppression

The result of the regression produces the optimal dimensions of the bounding boxes, the probabilities and the classification of the object. There are multiple bounding boxes covering the same area of the image. Therefore, multiple bounding boxes will overlap with each object detected. We need a way to find a single best bounding box for each object. The **non-max suppression algorithm** is just such an algorithm.

The non-max suppression algorithm proceeds in two steps:

1. The probabilities of objects in each of the boxes is computed.
2. Boxes with probabilities below some threshold, say 0.5, are eliminated from consideration.
3. For the remaining boxes:

- Select the remaining boxes with the highest probability.
- Compute the IoU for overlapping bounding boxes.
- Filter out bounding boxes with IoU above some threshold, such as 0.6.

4. Repeat step 3, until only one box per object remains.

Setting up for Exercises

In this assignment you will respond with short answers to a few questions about object detection algorithms. Only minimal coding is required.

You will start with the Object Detection Using Vision Transformer example Jupyter notebook. To run this notebook you will need a Google Colabratory account if you do not already have one. Log into your google account. Then, click the `View in Colab` tab to start the notebook in Colab. Alternatively, you may want to run this notebook locally, if you have the resources.

Overview of the example

The example in this notebook makes a number of simplifications. These simplifications make the code easier to understand and enable for faster model training. These simplifications include:

1. The model used only solves the problem of estimating the bounding box. All the objects are aircraft so no classificaiton is required. This reduces the model to a regression problem only.
2. In order to streamline the traning and evaluation of the transformer model a snall dataset based on the CalTec 101 dataset. The dataset is filtered so only aircraft images are used to train the model.

Modifications to the notebook

Before executing this notebook in Colab **you must makes some modifications as follows**:

1. Add a code cell containing the following to ensure the required package is available:

In [ ]:
```
!pip install -U tensorflow-addons
```

2. Next, you will update the last code cell in the notebook. The updated code will improves the evaluation of the model results by randomizing the images used for the visualization and IoU estimation. Replace the line of code at the start of the loop as shown below:

In [ ]:
```
# Compare results for 20 images in the test set
#for input_image in x_test[:10]:
np.random.seed(4567)
index = np.random.choice(x_test.shape[0],
                                    size=20,
                                    replace=False)
for input_image in x_test[index]:
```

3. Next, update normalization of the IoU calculation, in the second to last line of code in this cell, as follows:

In [ ]:
```
#print("mean_iou: " + str(mean_iou / len(x_test[:10])))
print("mean_iou: " + str(mean_iou / len(x_test[index])))
```

4. Finally, to evaluate the trajectory of the model traning, add the following code to a new cell at the end of the notebook:

In [ ]:
```
import matplotlib.pyplot as plt

def plot_hist(hist):
    plt.plot(hist.history["loss"], label="train")
    plt.plot(hist.history["val_loss"], label="validation")
    plt.title("Loss vs. Training Epoch")
    plt.ylabel("Loss")
    plt.xlabel("Epoch")
    plt.legend(loc="upper right")
    plt.show()

plot_hist(history)
```

Steps to submit notebook

Once you have successfully exectued your notebook in colab you must use `Save a copy in Drive` to save any of your work in your GoogleDrive account. You can then download the notebook from your GoogleDrive account and upload it into Canvas for submission.

Understanding the Code

Examine the code provided, read the discussion and complete the exerceises.

Download and prepare the dataset

The first major code block downloads the dataset and filters for only the aircraft images. The images are resized to the required 225×225 pixels.

Multilayer perceptron layer

A function is defined for the MLP layer used in each transformer head, in the code block. This layer has one fully connected layer and a dropout regularization layers.

Patch creation and encoding

The next three code cells define, display and sample and encode the patches.

1. The first of these cells defines a patch creation object with a method to divide an imput image into 49 patches, a 7×7 array.
2. A sample of one image and the resulting patches is displayued by the code in the next cell. Notice that the patches at the edge, in the horrizonatal direction, are a different dimension than those in the middle. This arrangement is required to give the patch and input image dimensions.
3. The third of these cells encodes the patch locations. Examine this code and notice how the positions are encoded for the patches.

> **Exercise 10-1:** Now, answer the following questions:
>
> 1. How does dividing the image into patches affect the range of attention in the images?
> 2. By what factor does dividing the image into these patches reduce the computational burden of computing attention, compared to the fully image of dimension 253×253?

> **Answer:**
>
> 1. Dividing the image into patches affects the range of attention in multiple ways. Among the main effects, this technique limits the area of attention to each patch reducing the complexity of the model, which allows to focus the attention on a smaller region of the image at a time. This also helps to improve the speed of processing and accuracy of the model by focusing on a relevant part of the image.
>
> 2. The *Attention is All you Need* paper states that separable convolutions decrease complexity to $O(k \cdot n \cdot d + n \cdot d^2)$. However, looking at the question from another perspective, I could say that the process is improved by the number of patches (49).

Define the model

The `create_vit_object_detector` funtion in the next code cell contains definiton of the actual model. In brief, the layers of this model, starting with the input are:

1. Layers that accept the image as input, divide the image into patches, and then encode the patches.
2. The transformer layers are defined in the for loop. These transformer layers define the backbone of the model that creates the feature map.

3. 5 layers that compute the desired output. These layers define the bounding box prediction head of the model. The last of these layers call the `mlp` function defined above.

Examine the details of this code, along with the comments, to understand each of these steps in more detail.

> **Exercise 10-2:** Answer the following questions about the model defined in the code cell:
>
> 1. Is this a pure transformer model and why?
> 2. How is the architecture used in SSD or YOLO type models fundamentally different from the model created here?
> 3. Are the transformer layers defined in the for loop consistent with the original Vaswani, et.al, 2017 model and why?
> 4. There are 5 lines of code following the for loop. In a few sentances, describe the overall purpose of these lines of code along with a description of the therory of operation.

> **Answers:**
>
> 1. No, this is not a pure transformer because it utilizes patches and a multi-layer perceptron (MLP) head in addition to the transformer blocks.
>
> 2. SSD and YOLO models are fundamentally different from the model created here because they use convolutional layers to extract features from the image. The model created here uses transformer blocks to extract features from the image.
>
> 3. No, the transformer layers defined in this notebook and the one in Vaswani, et.al, 2017 model are different. There are sevral differences. Among others, the encoders in both models are different. Also, the outputs of the last transformer block in this notebook are reshaped with layers.Flatten() and used as the image representation input to the classifier head.
>
> 4. These are the 5 lines that follow the for loop:

`layers.LayerNormalization(epsilon=1e-6)(encoded_patches)` : This layer normalizes the encoded patches received from the previous line. `attention_output = layers.MultiHeadAttention(num_heads=num_heads, key_dim=projection_dim, dropout=0.1)(x1, x1)` : This layer computes the attention output. Here, the model factors the output space of the self-attention layer into

a set of independent subspaces learned separately, where each subspace is called a *head*. `x2 = layers.Add()([attention_output, encoded_patches])`: This layer adds the attention output to the encoded patches. `x3 = layers.LayerNormalization(epsilon=1e-6)(x2)`: This layer normalizes x2, which has the attention output and the encoded patches. `x3 = mlp(x3, hidden_units=transformer_units, dropout_rate=0.1)`: The encoded input is passed through the MLP (multi-layer perceptron) layer. The MLP layer has one fully connected layer and a dropout regularization layer to predict 1 out of the K classes.

Train the model

The code in the next cell trains the model. The hyperparameters of the model and the training are defined. Then the model is trained for 40 epochs.

Notice there are two deviations from usual practice in traning transformer models:

1. The learning is supervised, with no unsupervised pre-training.
2. No data augmentation is employed.

Evaluate the model

The code in the last cell provided performs an evaluation on the model. This evaluation is limited by using only 20 test images. For each image the bounding box is predicted and the IoU is computed. The average IoU is printed. Finally, the images with the predicted bounding box and human-labled bounding box are ploted side by side.

> **Exercise 10-3:** Answer the following questions:
>
> 1. Examine the differences between the human-labeled bounding boxes and the predicted bounding boxes, along with considering the IoU value displayed. What do these metrics tell you about how well the model is predicting the bounding box. Your answer is necessarily going to be a bit subjective.
> 2. Is there a significant problem with scale for the objects in this particular in this dataset and how might this affect the model architecture?

> **Answers:**

> 1. The resulting IoU indicates that the model is computing the bounding box with an accuracy that ranges between 0.72 and 0.86 compared with the human bounding box, which is reasonable if we just want to get the main features of the fuselage. However, the model is missing

> the nose and the wheels in all cases. Something to note is that the model only executed 28 of the 100 epochs, so probably more training would improve the results. Also, even the human bounding boxes are not perfect because in some of them the wheels are excluded, or the tail is cut, or extra space is added. That means that given the training epochs and the labeled data, the model is not performing that bad.

> 2. I do not see a scaling problem. All airplanes have similar dimensions and the model is able to detect them, creating bounding boxes that enclose similar proportions of the fuselage.

> **Exercise 10-4** Finally, the code cell you have added displays the training and test loss vs. the traning epochs. What does this plot tell you about the convergence of the model training?

> **Answer:**

> The plot tells us that the model is converging right. The model learns a lot until the 5th epoch and then gets very stable, although continues learning a little bit until epoch 21 or 22. This is a good sign because it means that the model is not overfitting. The validation loss is also very stable after epoch 5, which confirms that the model is not overfitting.

In [ ]: