MMAI 5500 Assignment 2

DIY candy counter

In this assignment you will teach an object detector to count different types of candy. You will annotated images of candy and fine-tune an object detector adapting a Huggingface tutorial. The tutorial will have to be modified to work with your dataset (see below).

Data

Data labelling

You will label your own dataset using Label Studio. Install it on your local machine by following the instructions on Label Studio Quick Start.

Label the 10 images in `candy_images.zip` with the following 8 candy types:

```
{
  'Moon': 1,
  'Insect': 2,
  'Black_star': 3,
  'Grey_star': 4,
  'Unicorn_whole': 5,
  'Unicorn_head': 6,
  'Owl': 7,
  'Cat': 8
}
```

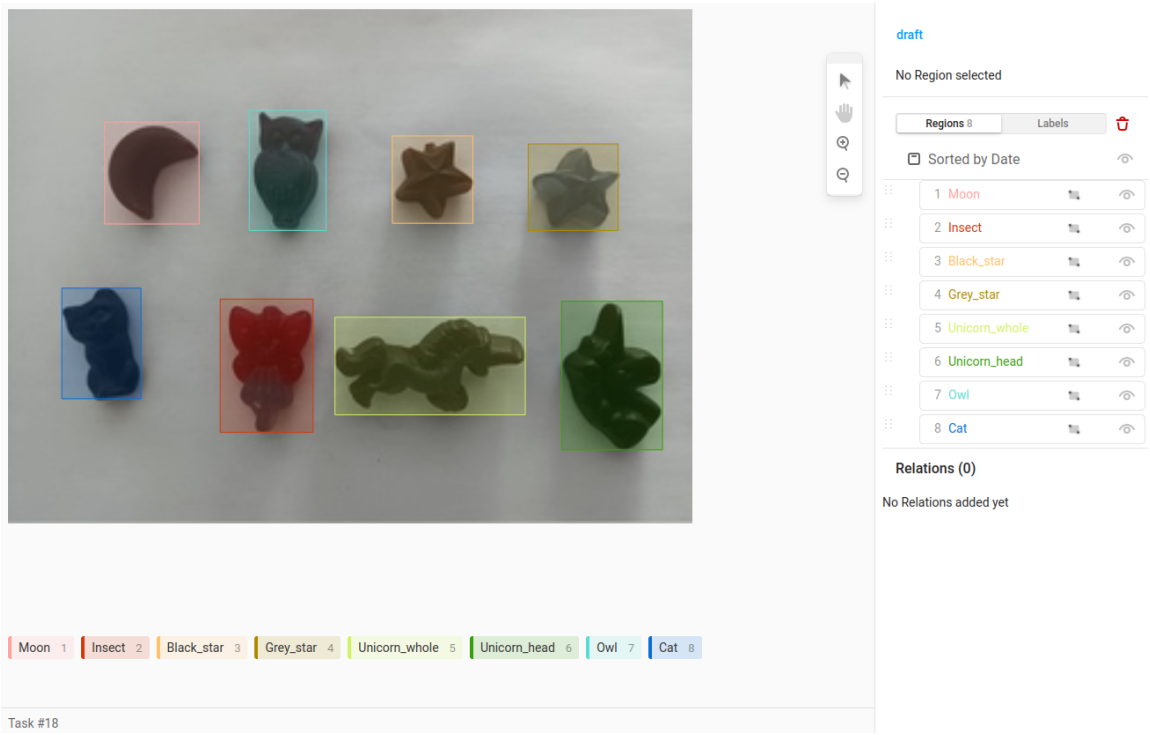
More labelling examples are included in the ZIP file `MMAI5500_Assignment2.zip`. Make the bounding boxes as tight around the objects as possible.

Export the images and annotations in the COCO json format. The export should return a ZIP archive containing a folder `images` and a file `result.json`. The former contains the images and the latter is a `json` file with the annotations (i.e. bounding boxes and class labels).

Load the data with Huggingface's `DatasetDict`

The annotated data exported from Label Studio cannot be directly loaded with Hugginface's `datasets.load_dataset()`. Below are some code snippets to help with the conversion.

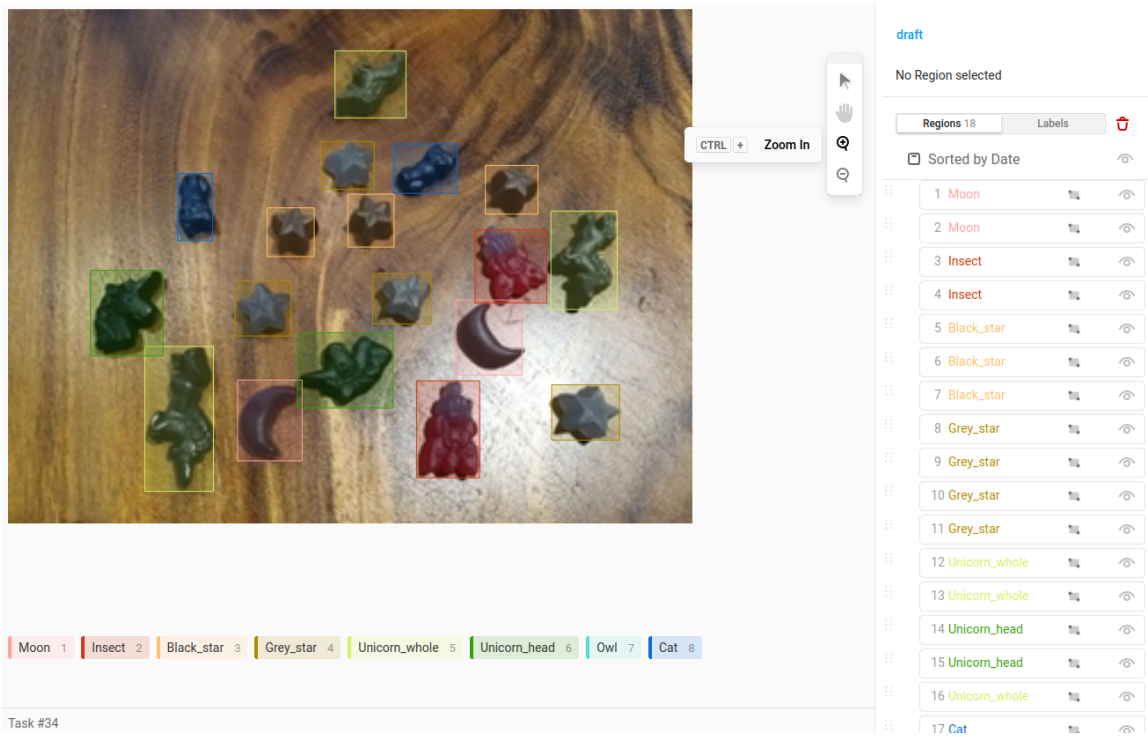
1. Read the COCO formatted annotations to a `dict`




```
import json
with open('result.json') as f:
  cocodata = json.load(f)
```

2. Convert the COCO formatted `dict` to a format readable by `dataset.load_dataset()`.

```
import os
# Store Huggingface formated data in a list
huggingdata = []
# Iterate through the images
for image in cocodata['images']:
  # Remove the image directory from the file name
  image['file_name'] = image['file_name'].split(os.path.sep)[-1]
  image['image_id'] = image['id']
  # Extend the image dict with bounding boxes and class labels
  image['objects'] = {'bbox': [], 'category': [], 'area': [], 'id': []}
  # Iterate through the annotations (bounding boxes and labels)
  for annot in cocodata['annotations']:
    # Check if the annotation matches the image
    if annot['image_id'] == image['id']:
      # Add the annotation
```




```
        image['objects']['bbox'].append(annot['bbox'])
        image['objects']['category'].append(annot['category_id'])
        image['objects']['area'].append(annot['area'])
        image['objects']['id'].append(annot['id'])
    # Append the image dict with annotations to the list
    huggingdata.append(image)
```

3. Write the Huggingface formatted annotations to a `json` file.

```
with open("metadata.jsonl", 'w') as f:
    for item in huggingdata:
        f.write(json.dumps(item) + "\n")
```

4. Follow Create an image dataset and the section Create an image dataset: object detection to organize the images and `metadata.jsonl` into the correct folder structure.

5. Read the data into a `DatasetDict`.

```
from datasets import load_dataset
# Assumes the data is stored in a folder called "data".
candy_data = load_dataset('imagefolder', data_dir="data")
```

To convert from ids to labels and back again (labels to ids) you can use this snippet:

```
id2label = {item['id']: item['name'] for item in cocodata['categories']}
label2id = {v: k for k, v in id2label.items()}
```

Save the model

Save the model using `trainer.save_model('candy_detector')` and downoad it from Colab.

Tips and tricks

Training

Your dataset is very small and much smaller than the `cppe5` dataset used in the tutorial. Thus, you will probably have to finetune your model for more epochs than in the tutorial. You can set the number of epochs in `TrainingArguments()` with the argument `num_train_epochs`. Make sure that the models learns, i.e. you see the training loss decreasing. `trainer.train()` will print the training loss according the the `logging_steps` set in the `TrainingArguments()`. If you set the number of `logging_steps=10` then the loss will be printed every 10 steps (batches).

Detection

Test your model by running inference/detection on some images. When you do this, you might not get any detections. A possible reason for this is that you set the detection threshold too high. In the tutorial they use `threshold=0.5` but this might be too high for your model. Try lowering the threshold.

With the pipeline you can set the threshold as follows:

```
obj_detector = pipeline("object-detection", model="candy_detector")
obj_detector(image, threshold=0.5)
```

Deliverables

You need to submit a notebook (`IPYNB`) and the trained model. The notebook should prepare and load the data, fine-tune the model, evaluate and save it. It should also have a method called `candy_counter(image)` that loads the fine-tuned model that takes as input a picture of candy (a Numpy array with the

same shape as the original images) and returns a dictionary with the counts of the different types of candies. `candy_counter(image)` should use the pipeline for inference. All the cells should be correctly executed, including a test image with predicted boundary boxes and saved in the notebook.

Example output:

```
In [6]: candy_counter(image)
Out[6]:
{'Moon': 5,
 'Insect': 0,
 'Black_star': 0,
 'Grey_star': 6,
 'Unicorn_whole': 2,
 'Unicorn_head': 1,
 'Owl': 0,
 'Cat': 4}
```

Grading

For full marks the submitted code needs to be bug free, include everything described under **Deliverables**, and the `candy_counter` has to have a mAP score above 0.5 on unseen test data.

Good luck!