# How to BubTFOD

* Adapted from Nicholas Renotte
* Uses Tensorflow Object Detection API
* Folder structure that works with contained Code

## (Create Artificial Images): create\_training\_images.py

Not part of the BubTFOD Set up, but essential for training a net

Creation of artificial bubble nucleation images

### What it does

* Random bubble instances are placed on a background showing a substrate
* Optional: perform random alterations (blur, rotate etc.)
* Storage of bounding box and mask of the image
* Optional: Save xml annotation file (containing info about placed bubbles) = necessary for training with BubTFOD

### What to set up

* Number of bubbles per image
* Number of images that should be generated
* Outdir: name of output directory (set to new name in order not to overwrite previously generated data)
* Xml\_bdbox: Set to TRUE for saving bounding box annotations in xml file
* Sigma\_rotate\_degree
* Sigma\_scaling\_fraction
* Highlight\_bubbles
* Blur\_mask

## Training and Detection

Object Detection (here: Bubbles) with a Deep Learning Network

Executes transfer learning with pretrained model from Tensorflow model zoo

### What it does

1. Set up of paths for pretrained and custom model
2. Load pre-trained Model (from tf model zoo)
3. Create Label Map

- Contains all categories that should be detected

- Here: category name “Bubble”; label id 1

1. Create TF records

* .record = binary storage format
* Convert images and their annotations into format for training/testing
* Storing data as sequence of binary strings has impact on training time
* (less space on disk, copy faster, read efficiently)

1. Copy model config (pipeline.config) to training folder

* pipeline.config defines architecture of model
* contains e.g. prebuilt data augmentation, max. number of allowed detections etc.
* see preset configurations in pipeline file in pre-trained model folder

1. Update config for transfer learning

* Change preset configurations in pipeline.config
* E.g. batch size, label map etc.

1. Train the model

* Set the number of training steps
* Print a command
* Run command in command prompt (in VS Code in included prompt possible)
* Event file (info/outcome of training) stored in model path – “Train”

1. Evaluate the model

* Calculate performance metrics
* Precision @ different IoUs and for different object sizes
* Recall @ different numbers of detection and for different object sizes
* Event file (info/outcome of test) stored in model path – “Eval”

1. Load model from checkpoint
2. Detect from an image
3. Freezing the graph
4. Conversion to TFJS
5. Zip and export Model

### What to Set up

* Prerequisite: Installed Tensorflow and dependent packages
* Name custom model (own name)
* Indicate url of pretrained model from tf model zoo (right click on desired model, copy link)
* Indicate proper name of pretrained model (part of copied url string)
* Modify pipeline.config
  + Pipeline\_config.model: indicate right model name
  + Set batch size
* Train the model:
  + Set number of training steps
  + Copy command into command prompt and run from there for being able so track the training
* Evaluate the model:
  + Print command, run from command prompt
  + Possible to see results with tensorboard (instructions in code)

## MetricPlots

Contains functions for plotting training and evaluation metrics

* Precision\_plot()
  + 6 Subplots that show:
    - mAP (mean average precision)
    - mAP @ 0.5 IOU
    - mAP @ 0.75 IOU
    - mAP for small/medium/large objects
  + input: dictionary with model names as keys; contains evaluation metrics
* Recall\_plot()
  + 6 Subplots that show:
    - AR (average recall) @ 1/10/100
    - AR @ 100 for small/large/medium objects
  + input: dictionary with model names as keys; contains evaluation metrics
* Plot\_total\_loss()
  + Plot loss curve from training
  + Input: df containing loss over time from (from training event file)
* Plot\_learningrate()
  + Plot learning rate from training
  + Input: df containing learningrate over time (from training event file)

## Model Evaluations

### What it does

* Gets properties from Train and Eval Event files of custom models
* Properties that can be seen with tensorboard (interactive interface)
* Train properties: Total loss, Detection loss, learning rate etc.
* Eval properties: Precisions, Recalls
* Uses Functions precision\_plot(), recall\_plot(), plot\_total\_loss(), and plot\_learningrate() from MetricPlots.py

1. Set\_paths()
2. Get\_event\_file()
3. Launch\_tensorboard()
4. Get\_evaluation\_metrics()
5. Save\_dict\_to\_excel()

### What to set up

* Indicate custom model names that you want to evaluate
* Set name of excel file containing Train\_Metrics and Eval\_Metrics
* Set name of precision plot
* Set name of recall plot
* Optional: launch tensorboard

## Models Separate Test

### What it does

Tests custom model on set of test images

Visualize Detections / Ground Truth bounding boxes

Calculate and plot bubble size distribution

Can produce the following plots/images:

1. Visualize the bounding boxes of predicted/detected bubbles in the original image
2. Visualize the bounding boxes of annotated bubbles in the original image
3. Visualize the bounding boxes of both predicted and annotated bubbles
4. Bubble size distribution: Histogram of the diameter; probability density (kernel density estimation – Gaussian) [for Prediction, Annotation, and the comparison of Prediction & Annotation]
5. Boxplot: Compare statistical properties of predicted and annotated bubble diameters
6. Plot: total number of bubbles over time

### What to set up

* Indicate name of custom model that should be tested
* Indicate (training) checkpoint of custom model (check model folder for latest checkpoint to use fully trained model)
* Indicate max. allowed detections in one image
* Indicate test\_path: set of images that model should be tested with
* Set Threshold “MIN\_SCORE\_THRESH”: only bounding boxes of bubbles with a detection score > threshold will be visualized in the plot
* Indicate the width and height of the image in millimeters

## TFODPaths

Contains function for getting the folder/path structure for a custom DL model

* Input: Name of custom model
* Output
  + Paths: All paths for custom model that fit to folder structure of BubTFOD
  + Files: Paths to files such as pipeline.config, labelmap, tf\_record\_script