Ary Naim

anaim@unm.edu

Step by Step guide for Collab Notebook "PUBLIC_yolov8_fire_and_smoke_retrain.ipynb"

Complete training pipeline to train YOLOv8 classifier for fire & smoke detection based on images.

Prerequisite Steps:

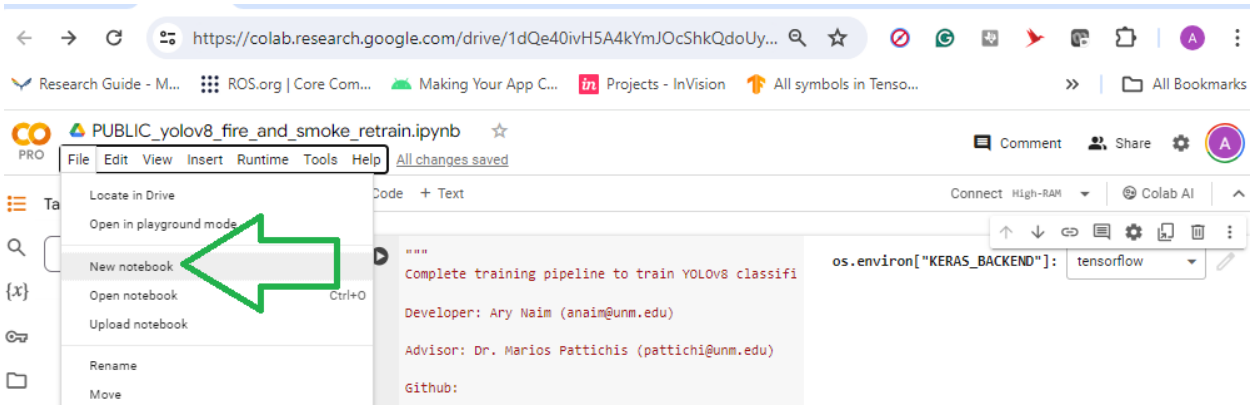
1. Create a Gmail account, in order to have a Google Drive for cloud storage.
2. Create a Google Collab account.
   This is online programming environment that gives you access to GPUs, CPUs, and can be shared with other people.

   Note: Google is not free after the 1st several hours and has both monthly cost and hourly cost for GPUs.

3. Navigate to "PUBLIC_yolov8_fire_and_smoke_retrain.ipynb" at

   https://colab.research.google.com/drive/1dQe40ivH5A4kYmJOcShkQdoUyl0sH5F0?usp=sharing

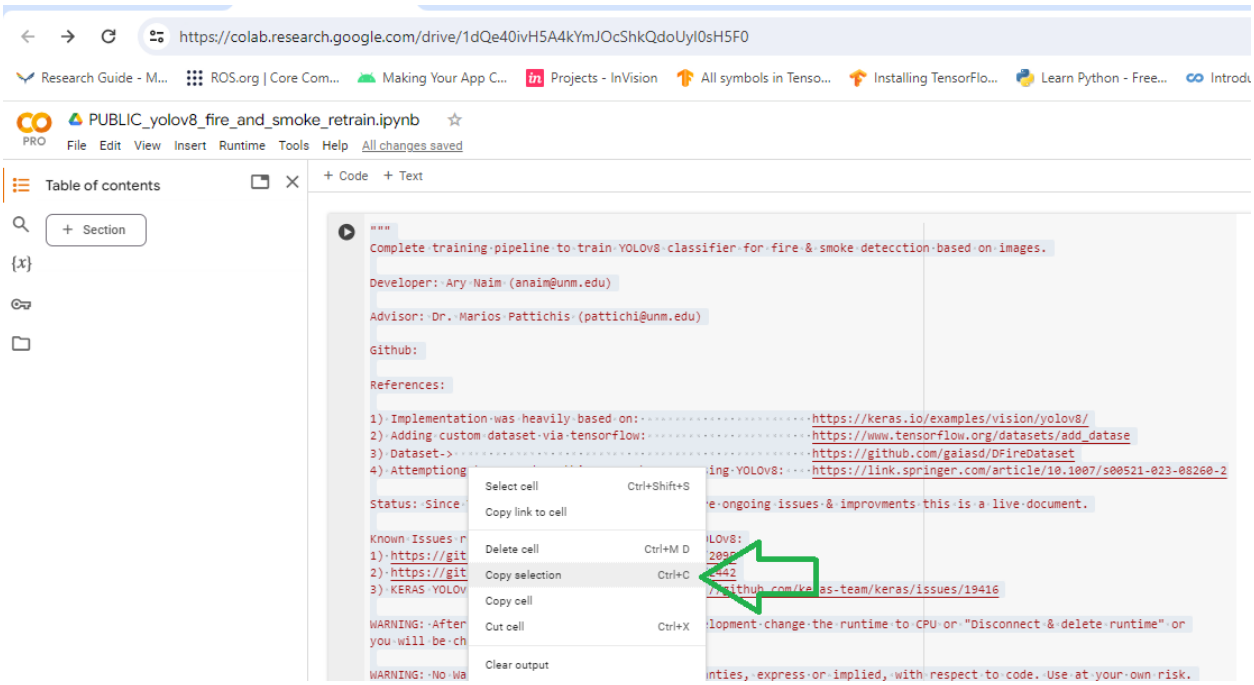
   This notebook can only be viewed or read therefore in order to make modifications create your own Notebook in Collab.



Let's start:

Step 0:

Copy all of the code from "PUBLIC_yolov8_fire_and_smoke_retrain.ipynb" into your own notebook.



Step 1:

Since we are going to run long running tasks we need a way to keep Google Collab alive even after hours of running. Even if you sign for Google Collab's "professional" version the browser will timeout after ~30 min of inactivity.

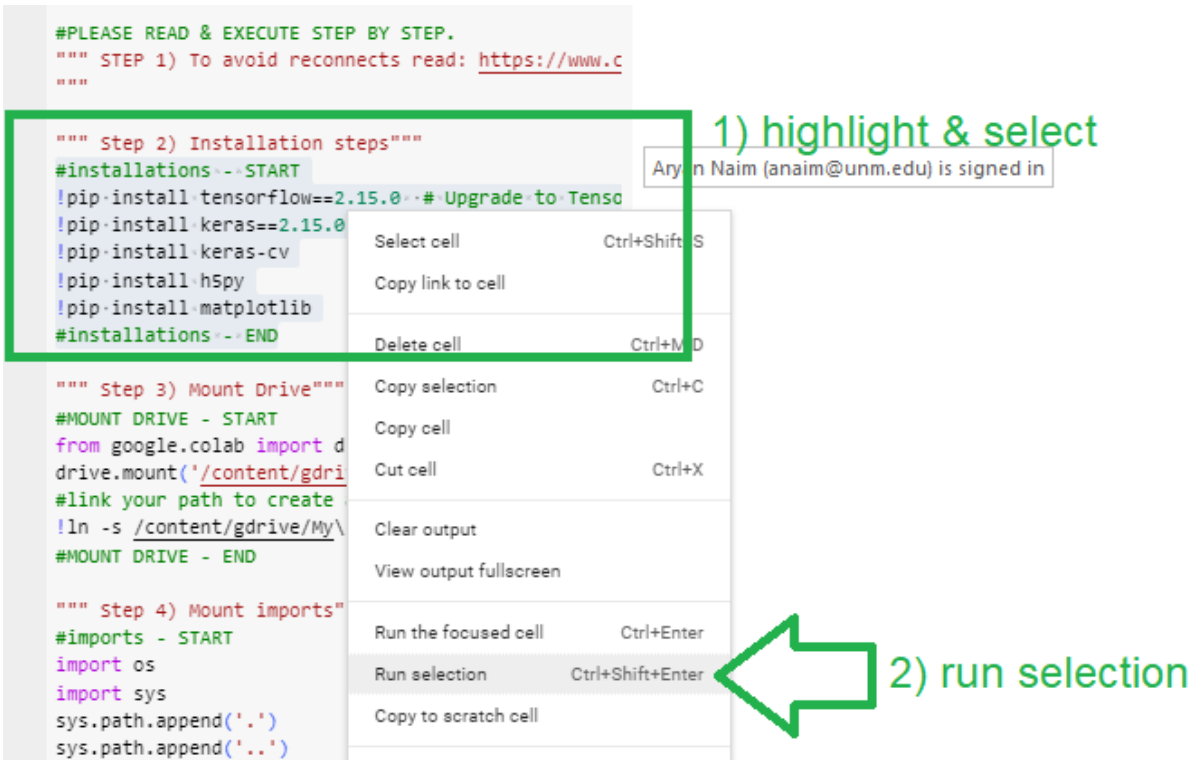
Do this hack to keep your browser alive:

```
https://www.codeease.net/programming/javascript/keep-colab-from-disconnecting
```

Step 2:

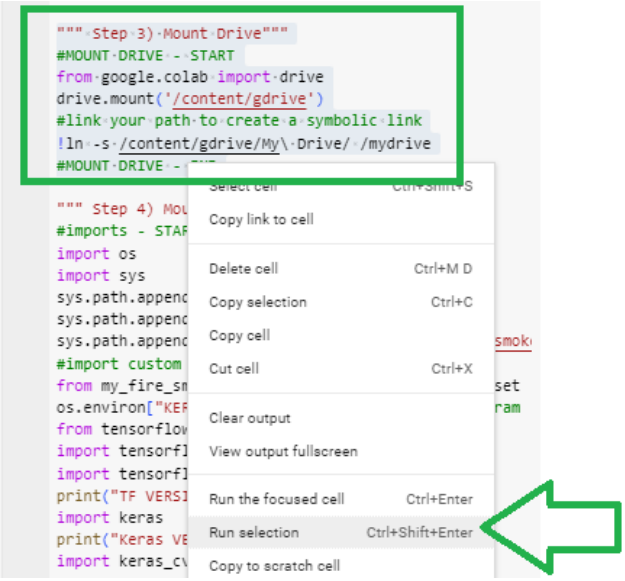
Based on the Python code in “PUBLIC_yolov8_fire_and_smoke_retrain.ipynb” install the required Python libraries.

High & run the installation only:

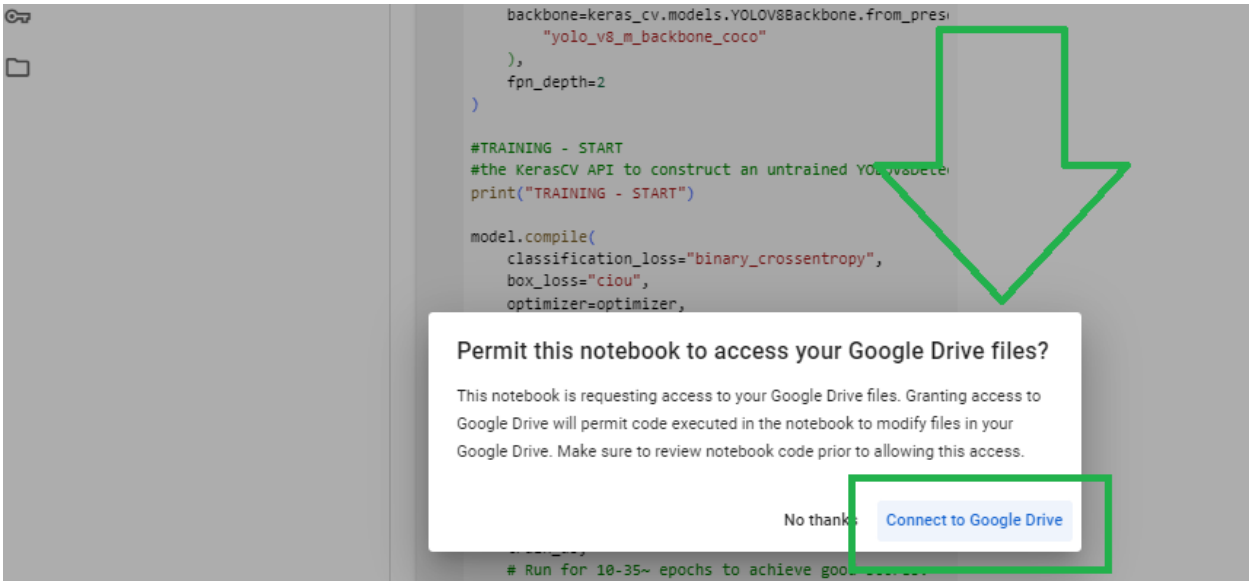


Step 3:

Mount your Google Drive

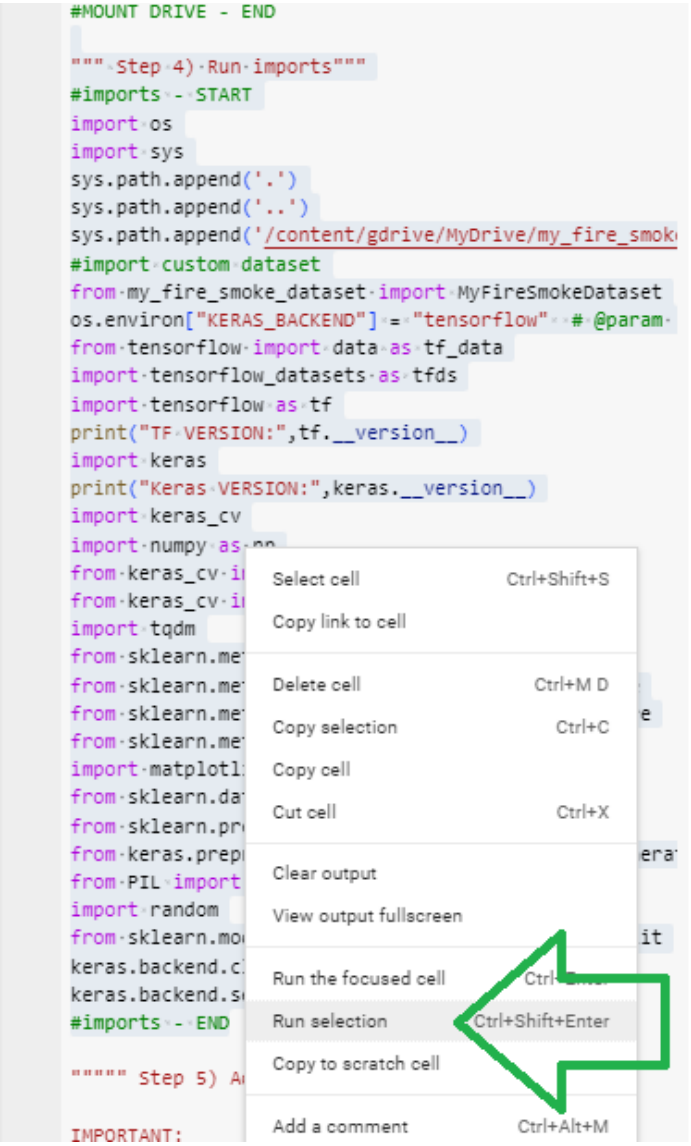


You will be asked for permissions via several dialog boxes. Approve them.



Continue to step 4 on the next page.

Step 4: Import the required libraries.

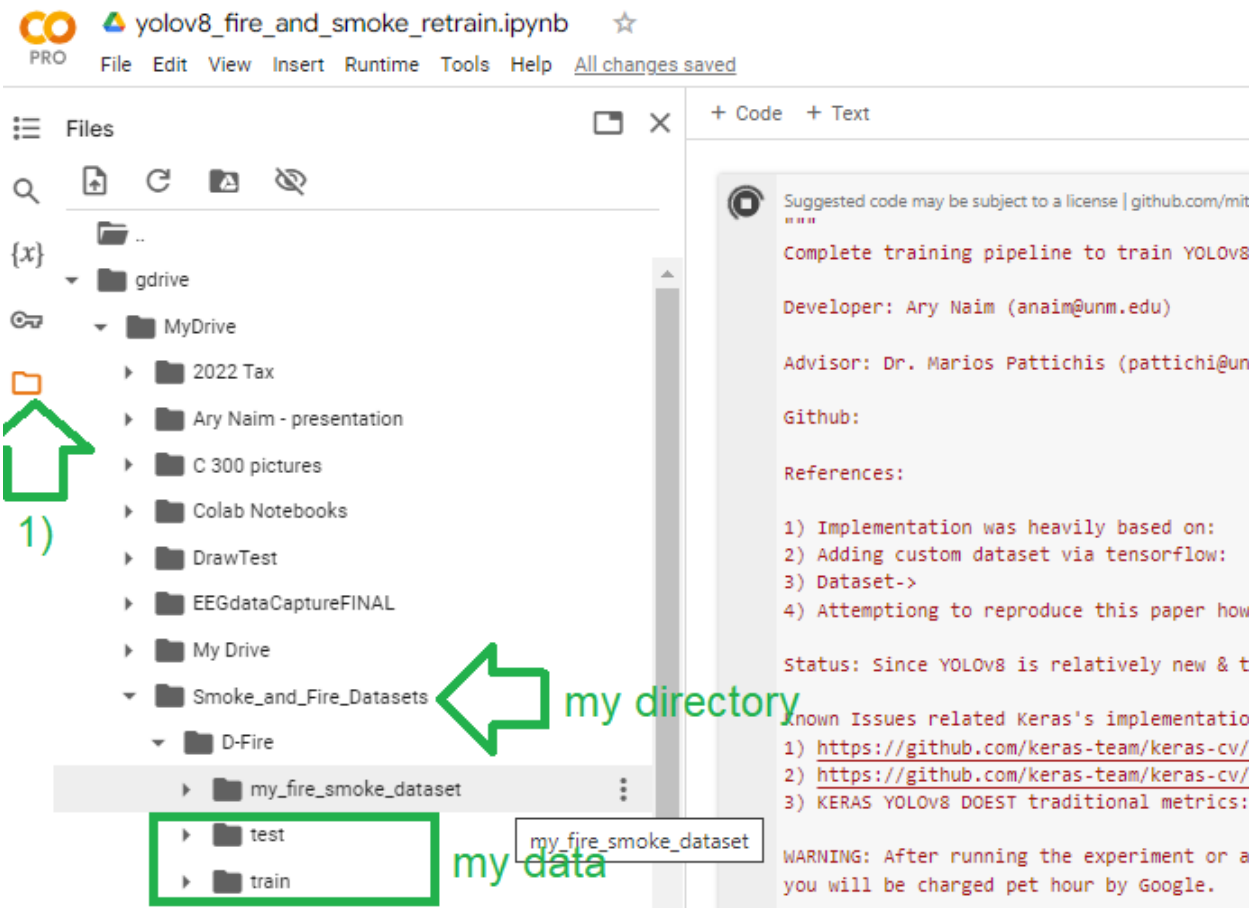


There should be no errors. If there are any installation or import errors do not proceed until all of the errors are resolved.

Step 5) ADVANCED - Optional for Custom Datasets.

If you are using dataset from Keras or just want to run the Collab notebook as is, you can skip this step #5 & go to step 6.

a) Create a folder in Google drive to upload your data & write a custom Python ()

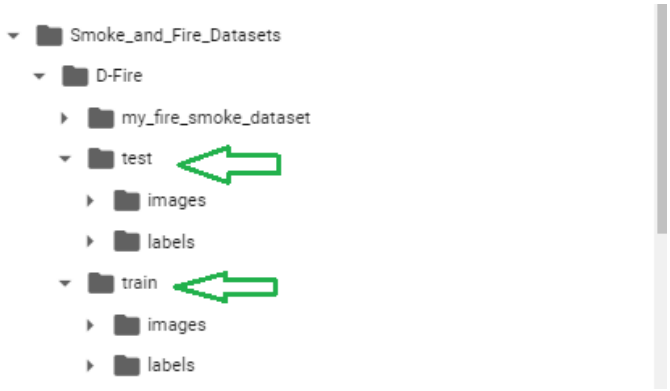


The path to my data is. Your can be different.

/content/gdrive/MyDrive/Smoke_and_Fire_Datasets

Darknet/YOLOv1-4 data format:

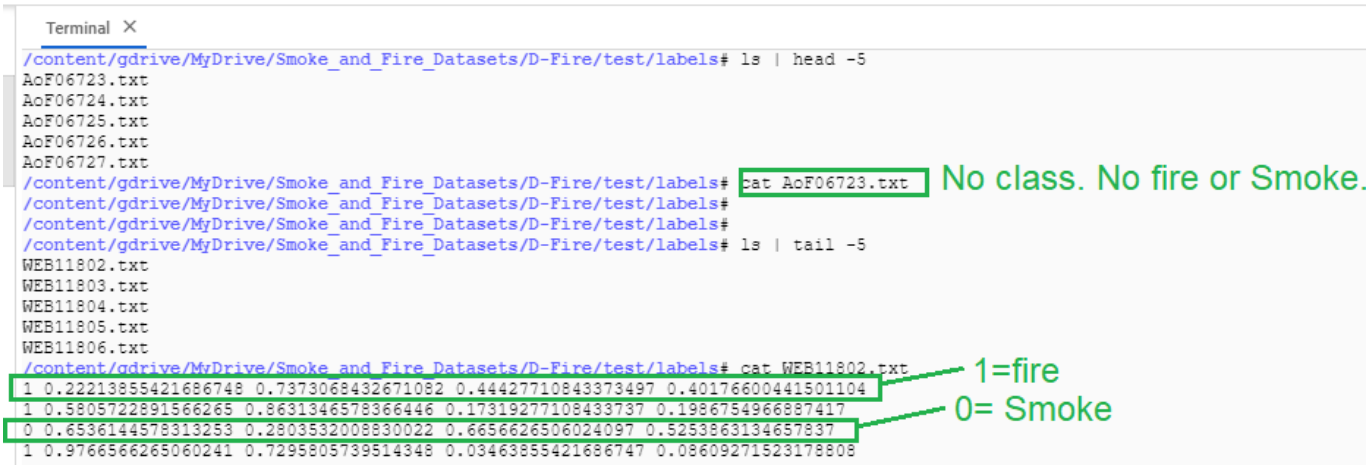
In this specific example, the original dataset was preprocessed to be in “test” & “train” folders and each of those folders has a “image” and “labels” subfolder:



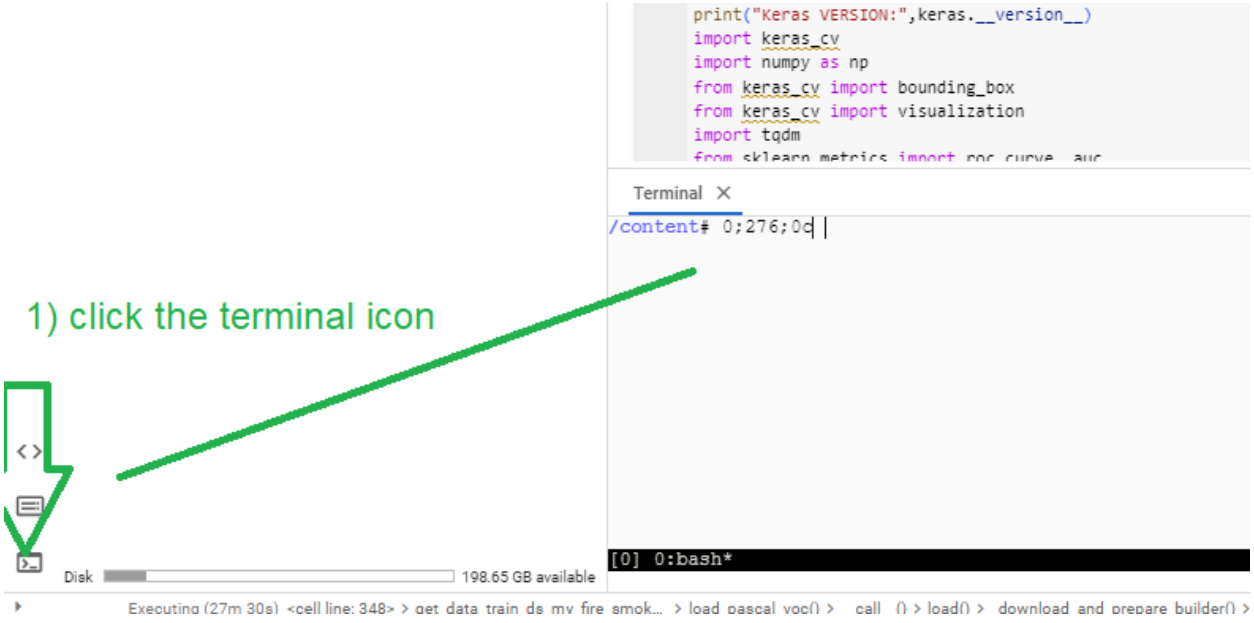
YOLO versions v1 to v4 were trained using Darknet neural network framework (https://pjreddie.com/darknet/) therefore a lot order datasets have the above format of test/images, test/labels, train/images, and train/labels.

The “images” folder have images files with corresponding text label files under “labels” folder, with same file name.

A closer inspection of the labels shows files that have no data indicating no classification and label files with label data in the format <class,x,y,width,height>. See below:



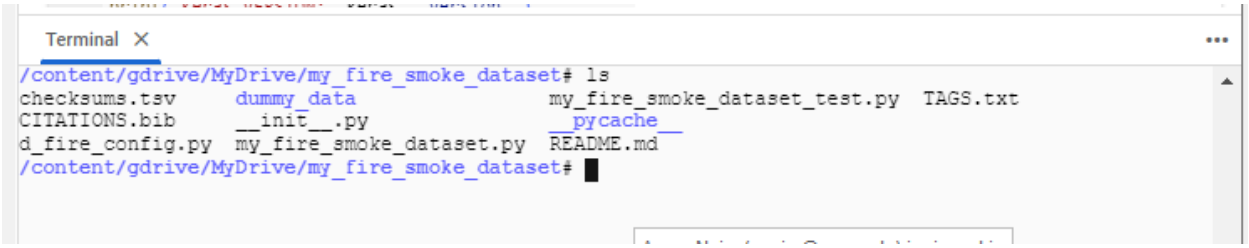
b) Open the online browser terminal:



c) Navigate to were you uploaded your data

For example:

cd /content/gdrive/MyDrive/my_fire_smoke_dataset

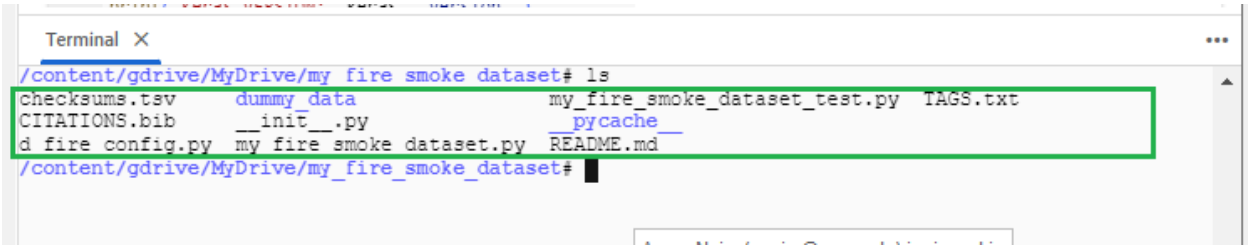


All the files you see here are auto generated & you see them after executing the next command.

d) In the same path execute this command:

```
tfds new <THE_NAME_OF_YOUR_DATASET>
```

You should see bunch of files & folders appear:



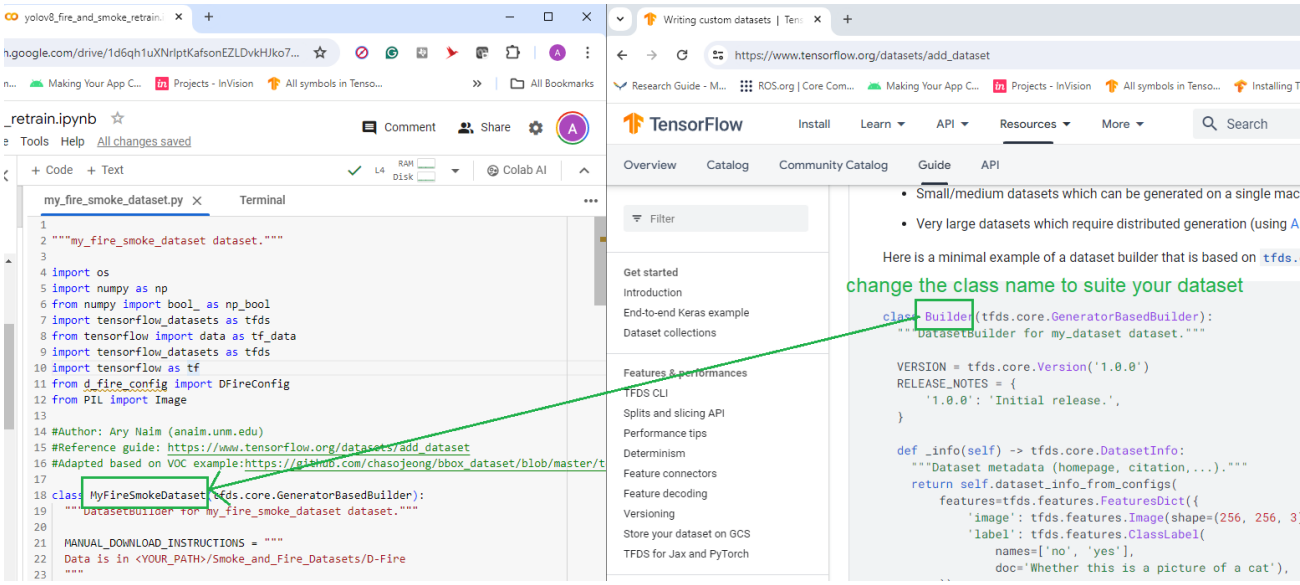
(Writing custom datasets) Follow this guide step by step:

READ carefully & then come back to this guide:
https://www.tensorflow.org/datasets/add_dataset

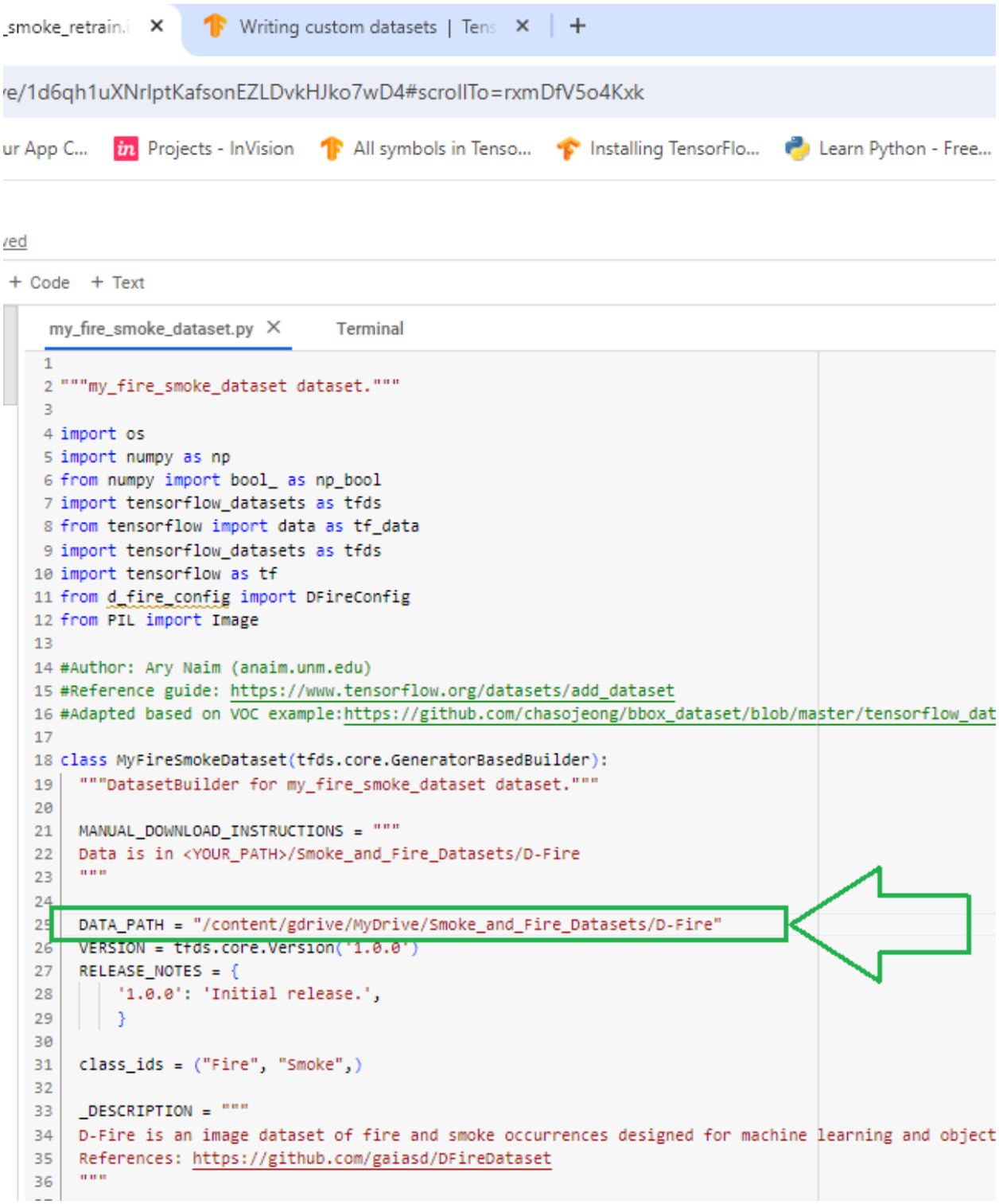
e) Open the autogenerated “my_dataset_dataset_builder.py” file in one browser & keep the (https://www.tensorflow.org/datasets/add_dataset) guide open in another window as a guide.

Your custom data handling class must implement tfds.core.DatasetBuilder

Change the class name to match your dataset. Our dataset has fire & smoke images there we named it MyFireSmokeDataset:



f) Since we are load the data from folder rather from a web service, then hard code the path to your data:



g) Change the classes to match your datasets labels:

```
class_ids = ("Fire", "Smoke",)
```

h) Implement the functions minimum required functions that are inherited from tfds.core.GeneratorBasedBuilder. Therefore you must implement at the minimum:

_info()
_split_generators()
_generate_examples():
_generate_example():

For example, for _info, this was our implementation:

```
"""
function: _info(self)
Input: None
Output: tfds.core.DatasetInfo
Description:
Returns tfds.core.DatasetInfo which is the class that describes our
dataset.
"""
def _info(self):
    return tfds.core.DatasetInfo(
        builder=self,
        description=self._DESCRIPTION,
        features=tfds.features.FeaturesDict({
            "image": tfds.features.Image(),
            "image/filename": tfds.features.Text(),
            "objects": tfds.features.Sequence({
                "label": tfds.features.ClassLabel(names=self.class_ids),
                "bbox": tfds.features.BBoxFeature(),
                "is_truncated": np_bool,
                "is_difficult": np_bool,
            }),
            "labels": tfds.features.Sequence(
                tfds.features.ClassLabel(names=self.class_ids)
            ),
            "labels_no_difficult": tfds.features.Sequence(
                tfds.features.ClassLabel(names=self.class_ids)
            ),
        }),
        homepage="https://github.com/gaiasd/DFireDataset",
        citation="""Pedro Vinícius Almeida Borges de Venâncio,
        Adriano Chaves Lisboa, Adriano Vilela Barbosa:
        An automatic fire detection system based on deep
        convolutional neural networks for low-power,
        resource-constrained devices.
        In: Neural Computing and Applications, 2022""",
    )
```

For the function _split_generators()

```
def _split_generators(self, dl_manager: tfds.download.DownloadManager):
  """Returns SplitGenerators."""
  extracted_path = (self.DATA_PATH)
  return {
    'train': self._generate_examples(
        images_path=str(extracted_path)+str("/train/images"),
        labels_path=str(extracted_path)+("/train/labels"),
    ),
    'test': self._generate_examples(
        images_path=str(extracted_path)+str("/test/images"),
        labels_path=str(extracted_path)+("/test/labels"),
    ),
}
```

For the function _generate_examples():

```
def _generate_examples(self, images_path,labels_path):
    """Yields examples."""
    image_files = []
    labels_files = []
    # Yields (key, example) tuples from the dataset
    #1) step 1 - get images & labels (which have class & bounding boxes )
    image_files = self.get_jpeg_files(images_path)
    labels_files = self.get_txt_files(labels_path)
    image_files = image_files[0:500]
    labels_files = labels_files[0:500]
    #2) for each image & label get the files
    for img_f, lbl_f in zip(image_files,labels_files):
      #return the image bytes & bounding bix info
      filename = os.path.basename(img_f)
      image, bounding_boxes = self.load_image_bounding_box(img_f,lbl_f,DFireConfig.image_size_1)
      if bounding_boxes is not None:
        print("_generate_examples(), DATA:",bounding_boxes)
        yield filename, self._generate_example(img_f,bounding_boxes,bounding_boxes["class"])
      else:
        continue
```

limit for testing.Remove in production.

For the function _generate_example:

```
def _generate_example(self,image_filepath,bbox,label):
  objects = []
  labels = []
  labels_no_difficult = []
  label = bbox["class"]
  x = bbox["x"]
  y = bbox["y"]
  w = bbox["width"]
  h = bbox["height"]
  #convert from YOLO to relative BBox
  image_w, image_h = self.get_image_dimensions(image_filepath)
  yolo_box = (x,y,w,h)
  pixel_coords = self.yolo2pixel((image_w, image_h),[x,y,w,h])
  x1 = pixel_coords[0]/image_w
  y1 = pixel_coords[1]/image_h
  x2 = pixel_coords[2]/image_w
  y2 = pixel_coords[3]/image_h
  if x2 > 1:
    x2 = 1.0
  if y2 > 1:
    y2 = 1.0
  if label == "0":
    label = self.class_ids[0]
  if label == "1":
    label = self.class_ids[1]
    #References: https://www.tensorflow.org/datasets/api_docs/python/tfds/features/BBox
  objects.append({
        "label": label,
        "bbox": tfds.features.BBox(x1,y1,x2,y2),
        "is_truncated": False,
        "is_difficult": False,
    })
  return {
      "image": image_filepath,
      "image/filename": os.path.basename(image_filepath),
      "objects": objects,
      "labels": np.array([label]),
      "labels_no_difficult": labels_no_difficult,
  }
```

yolo related

i) After the implementation is complete then go the terminal & type

```
tfds build --register_checksums
```

This may take a while depending on the size of your dataset. It tool ~10 min for a dataset of size 500 & many hours for dataset of size 5000.

There should be no errors.

Step 6) Lets go back to the actual experiment “PUBLIC_yolov8_fire_and_smoke_retrain.ipynb”.

If you chose to do the Step 5 then make sure you can import your custom Python script at the top of the imports.

```
""" Step 4) Mount imports"""
#imports - START
import os
import sys
sys.path.append('.')
sys.path.append('..')
sys.path.append('/content/gdrive/MyDrive/my_fire_smoke_dataset') #You
#import custom dataset
from my_fire_smoke_dataset import MyFireSmokeDataset
os.environ["KERAS_BACKEND"] = "tensorflow"  # @param ["tensorflow", "
from tensorflow import data as tf_data
import tensorflow_datasets as tfds
```

Else run the section associated with declaring variables & functions.

```
"""step 6) declare variables & functions"""

#VARIABLES.
#IMPORTANT: change the labels here to reflect the actual labels you have based on your dataset.
class_ids = [
    "Fire",
     "Smoke",
]

image_size_1 = (640,640)
image_size_2 = (320,320)
image_size_3 = (160,160)

class_mapping = dict(zip(range(len(class_ids)), class_ids))


# FUNCTION DECLARATIONS - START
def visualize_dataset(inputs, value_range, rows, cols, bounding_box_format):
    inputs = next(iter(inputs.take(1)))
```

Step 7) Start the experiment & load data

step 7) Start the experiment & load data

Noe that we have imported all the required imports and declared all the required functions. Lets run the experiment.

**YOU NEED A GPU.* Makesure to switch the runtime to a GPU that you can afford (start with L4). You may need re-run previous steps as the runtime is restarted.

```
"""step 7) Start the experiment & load data"""
#PARAMATERS
BATCH_SIZE = 4
#hyperparameters - START
#hyperparameters
LEARNING_RATE = 0.005  #try 0.001 to 0.005
EPOCHS = 50 # 50 is not enough, the original paper ran epoches frpm 5,000 to 30,000
MOMENTUM = 0.9
#hyperparameters - END

#GET DATA
print("GET DATA - START ")
#original
```

You should start seeing the dataset being loaded. This will take a while even though we are only using 500 images which is not enough for building an accurate model.

```
GET DATA - START
Downloading and preparing dataset Unknown size (download: Unknown size, generated: Unknown size, total: Unknown size) to /root/tensorflow_
Generating splits...:   0%  0/2 [00:00<?, ? splits/s]
Generating train examples...:  267/? [13:12<00:00,  1.51s/ examples]
bounding_box: {'class': 'Fire', 'x': 0.4852941176470588, 'y': 0.25416666666666665, 'width': 0.8823529411764706, 'height': 0.45833333333333
_generate_examples(), DATA: {'class': 'Fire', 'x': 0.4852941176470588, 'y': 0.25416666666666665, 'width': 0.8823529411764706, 'height': 0.
bounding_box: {'class': 'Fire', 'x': 0.4115000000000003, 'y': 0.33066666666666666, 'width': 0.8130000000000001, 'height': 0.64533333333333
_generate_examples(), DATA: {'class': 'Fire', 'x': 0.4115000000000003, 'y': 0.33066666666666666, 'width': 0.8130000000000001, 'height': 0
bounding_box: {'class': 'Fire', 'x': 0.5024509803921569, 'y': 0.2, 'width': 0.8970588235294118, 'height': 0.3500000000000003}
_generate_examples(), DATA: {'class': 'Fire', 'x': 0.5024509803921569, 'y': 0.2, 'width': 0.8970588235294118, 'height': 0.350000000000000
bounding_box: {'class': 'Fire', 'x': 0.3802931596091205, 'y': 0.2305194805194805, 'width': 0.7312703583061889, 'height': 0.430735930735930
_generate_examples(), DATA: {'class': 'Fire', 'x': 0.3802931596091205, 'y': 0.2305194805194805, 'width': 0.7312703583061889, 'height': 0.4
bounding_box: {'class': 'Fire', 'x': 0.546875, 'y': 0.4847222222222222, 'width': 0.246875, 'height': 0.25277777777777777}
_generate_examples(), DATA: {'class': 'Fire', 'x': 0.546875, 'y': 0.4847222222222222, 'width': 0.246875, 'height': 0.25277777777777777}
bounding_box: {'class': 'Fire', 'x': 0.46718750000000003, 'y': 0.49444444444444446, 'width': 0.15937500000000002, 'height': 0.194444444444
_generate_examples(), DATA: {'class': 'Fire', 'x': 0.46718750000000003, 'y': 0.49444444444444446, 'width': 0.15937500000000002, 'height':
bounding_box: {'class': 'Fire', 'x': 0.45625000000000004, 'y': 0.5458333333333334, 'width': 0.096875, 'height': 0.13611111111111113}
_generate_examples(), DATA: {'class': 'Fire', 'x': 0.45625000000000004, 'y': 0.5458333333333334, 'width': 0.096875, 'height': 0.1361111111
bounding_box: {'class': 'Fire', 'x': 0.6078125000000001, 'y': 0.3875, 'width': 0.371875, 'height': 0.2638888888888889}
```

If you want to load the entire 20,000 images open the Python file title "my_fire_smoke_dataset.py" and make an update in the function:

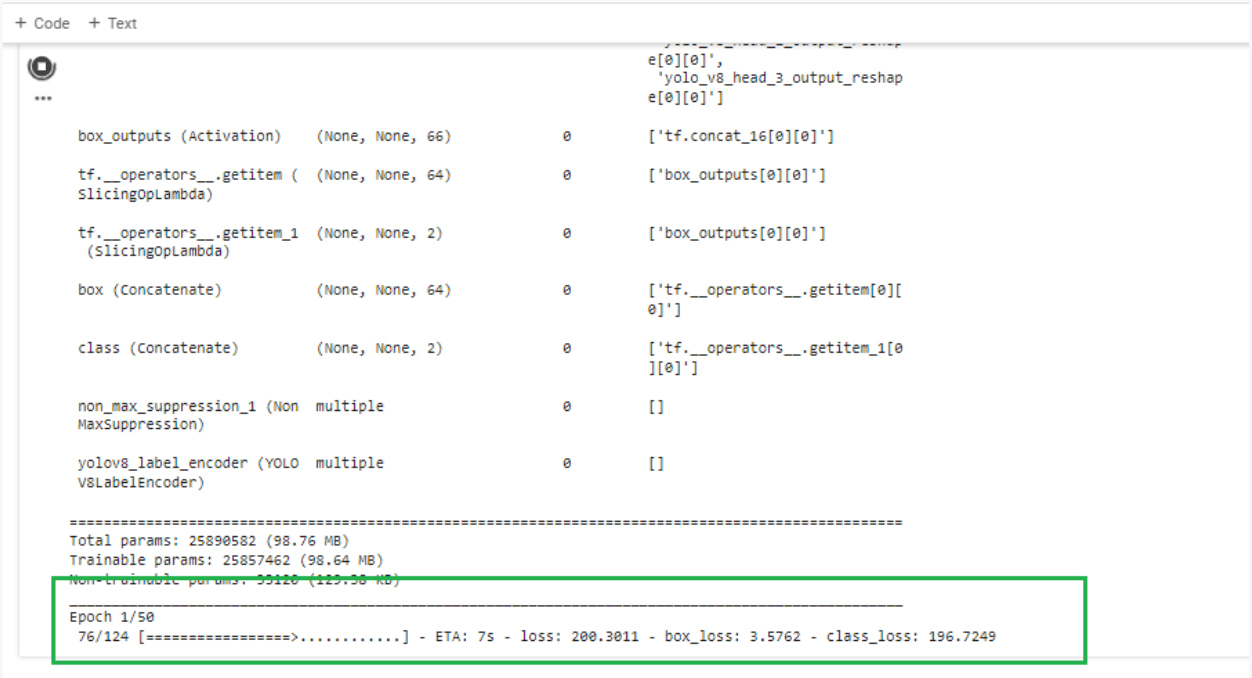
Terminal | my_fire_smoke_dataset.py

```
    return {
      'train': self._generate_examples(
          images_path=str(extracted_path)+str("/train/images"),
          labels_path=str(extracted_path)+("/train/labels"),
      ),
      'test': self._generate_examples(
          images_path=str(extracted_path)+str("/test/images"),
          labels_path=str(extracted_path)+("/test/labels"),
      ),
  }

  def _generate_examples(self, images_path,labels_path):
      """Yields examples."""
      image_files = []
      labels_files = []
      # Yields (key, example) tuples from the dataset
      #1) step 1 - get images & labels (which have class & bounding boxes )
      image_files = self.get_jpeg_files(images_path)
      labels_files = self.get_txt_files(labels_path)
      image_files = image_files[0:500]
      labels_files = labels_files[0:500]
      #2) for each image & label get the files
      for img_f, lbl_f in zip(image_files,labels_files):
        #return the image bytes & bounding bix info
        filename = os.path.basename(img_f)
        image, bounding_boxes = self.load_image_bounding_box(img_f,lbl_f,DFireConfig.image_size_1)
        if bounding_boxes is not None:
          print("_generate_examples(), DATA:",bounding_boxes)
          yield filename, self._generate_example(img_f,bounding_boxes,bounding_boxes["class"])
        else:
          continue

```

change or remove this.

Now you should see training results such as epoch, some metrics per each epoch, and visualizations.

You can also see the model training per each epoch:



Step 8) **Inference**

Inference - Lets load & use the model

```
[13] #INFERENCE - Lets use the mode

     loaded_model = keras.saving.load_model(model_path)
     model = loaded_model
     if model is not None:
         print("Model loaded from disk")
     else:
         print("Failed to load model from disk")

     if model is not None:
         model.prediction_decoder = keras_cv.layers.NonMaxSuppression(
         bounding_box_format="xyxy",
         from_logits=True,
         iou_threshold=0.5,
         confidence_threshold=0.75,
     )

     #LOAD MODEL - END
     #INFERENCE
     visualization_ds = eval_ds.unbatch()
     visualization_ds = visualization_ds.ragged_batch(16)
     images, y_true = next(iter(visualization_ds.take(1)))
     y_pred = model.predict(images)
```

DONE .