Face Mask Detection Live ....

@Author Saurabh Kumar @Date : March_22

```
pwd
```

```
'E:\\DataScience\\Tensorflow_Object_detection\\RealTimeObjectDetection'
```

```
import os
os.listdir()
```

```
['.git',
 '.ipynb_checkpoints',
 'Face_Mask_Detection_Live.ipynb',
 'Tensorflow',
 'Tutorial.ipynb']
```

```
path_MAIN ="E:\\DataScience\\Tensorflow_Object_detection\\RealTimeObjectDetection'
path_tensorflow =path_MAIN+"Tensorflow"
os.listdir(path_tensorflow)
```

```
['labelImg', 'models', 'scripts', 'workspace']
```

```
path_wrkspc ="E:\\DataScience\\Tensorflow_Object_detection\\RealTimeObjectDetection\\Tensorflow\\workspace'
os.listdir(path_wrkspc)
```

```
['annotations', 'images', 'models', 'pre-trained-models']
```

```
path_img ="E:\\DataScience\\Tensorflow_Object_detection\\RealTimeObjectDetection\\Tensorflow\\workspace\\images'
os.listdir(path_img)
```

```
['Mask', 'No_Mask']
```

Setup paths

```
WORKSPACE_PATH = 'Tensorflow\workspace'
SCRIPTS_PATH = 'Tensorflow\scripts'
APIMODEL_PATH = 'Tensorflow\models'
ANNOTATION_PATH = WORKSPACE_PATH+'\annotations'
IMAGE_PATH = WORKSPACE_PATH+'\images'
MODEL_PATH = WORKSPACE_PATH+'\models'
PRETRAINED_MODEL_PATH = WORKSPACE_PATH+'\pre-trained-models'
CONFIG_PATH = MODEL_PATH+'\my_ssd_mobnet\pipeline.config'
CHECKPOINT_PATH = MODEL_PATH+'\my_ssd_mobnet\'
```

Label Map

```
labels = [{'name':'Mask','id':1},{'name':'No-Mask','id':2}]
```

```
labels
```

```
[{'name': 'Mask', 'id': 1}, {'name': 'No-Mask', 'id': 2}]
```

```
for label in labels:
    print('\tname:\'{}\'\n'.format(label['name']))
    print('\tid:{}\n'.format(label['id']))

    name:'Mask'

    id:1

    name:'No-Mask'

    id:2
```

```
with open(ANNOTATION_PATH+'\label_map.pbtxt', 'w') as f:
    for label in labels:
        f.write('item {\n')
        f.write('\tname:\'{}\'\n'.format(label['name']))
        f.write('\tid:{}\n'.format(label['id']))
        f.write('}\n')
```

TF record

```
!python {SCRIPTS_PATH + '/generate_tfrecord.py'} -x {IMAGE_PATH + '/train'} -l {ANNOTATION_PATH + '/label_map.pbtxt'} -o {ANNOTATION_PATH + '/train.record'}
!python {SCRIPTS_PATH + '/generate_tfrecord.py'} -x{IMAGE_PATH + '/test'} -l {ANNOTATION_PATH + '/label_map.pbtxt'} -o {ANNOTATION_PATH + '/test.record'}
```

```
Successfully created the TFRecord file: Tensorflow\workspace/annotations/train.record
Successfully created the TFRecord file: Tensorflow\workspace/annotations/test.record
```

```
!cd Tensorflow && git clone https://github.com/tensorflow/models
```

```
fatal: destination path 'models' already exists and is not an empty directory.
```

```
CUSTOM_MODEL_NAME = 'my_ssd_mobnet'
```

```
!mkdir {'Tensorflow\workspace\models\'+CUSTOM_MODEL_NAME}
!cp {PRETRAINED_MODEL_PATH+'/ssd_mobilenet_v2_fpnlite_320x320_coco17_tpu-8/pipeline.config'} {MODEL_PATH+'/'+CUSTOM_MODEL_NAME}
```

```
A subdirectory or file Tensorflow\workspace\models\my_ssd_mobnet already exists.
```

```
import tensorflow as tf
from object_detection.utils import config_util
from object_detection.protos import pipeline_pb2
from google.protobuf import text_format
```

```
tf.__version__
```

```
'2.5.0'
```

```
CONFIG_PATH = MODEL_PATH+'/'+CUSTOM_MODEL_NAME+'/pipeline.config'
```

```
config = config_util.get_configs_from_pipeline_file(CONFIG_PATH)
```

```
config
```

```
{'model': ssd {
   num_classes: 2
   image_resizer {
     fixed_shape_resizer {
       height: 320
       width: 320
     }
   }
   feature_extractor {
     type: "ssd_mobilenet_v2_fpn_keras"
     depth_multiplier: 1.0
     min_depth: 16
     conv_hyperparams {
       regularizer {
         l2_regularizer {
           weight: 0.0999999895151007e-05
         }
       }
       initializer {
         random_normal_initializer {
           mean: 0.0
           stddev: 0.0099999997764825
         }
       }
       activation: RELU_6
       batch_norm {
         decay: 0.9969999719619165
         scale: true
         epsilon: 0.0010000000474974513
       }
     }
     use_depthwise: true
     override_base_feature_extractor_hyperparams: true
     fpn {
       min_level: 3
       max_level: 7
       additional_layer_depth: 128
     }
   }
   box_coder {
     faster_rcnn_box_coder {
       y_scale: 10.0
       x_scale: 10.0
       height_scale: 5.0
       width_scale: 5.0
     }
   }
   matcher {
     argmax_matcher {
       matched_threshold: 0.5
       unmatched_threshold: 0.5
       ignore_thresholds: false
       negatives_lower_than_unmatched: true
       force_match_for_each_row: true
       use_matmul_gather: true
     }
   }
   similarity_calculator {
     iou_similarity {
     }
   }
   box_predictor {
     weight_shared_convolutional_box_predictor {
       conv_hyperparams {
         regularizer {
           l2_regularizer {
             weight: 0.0999999895151007e-05
           }
         }
         initializer {
           random_normal_initializer {
             mean: 0.0
             stddev: 0.0099999997764825
           }
         }
         activation: RELU_6
         batch_norm {
           decay: 0.9969999719619165
           scale: true
           epsilon: 0.0010000000474974513
         }
       }
       depth: 128
       num_layers_before_predictor: 4
       kernel_size: 3
       class_prediction_bias_init: -4.5999999046325684
       share_prediction_tower: true
       use_depthwise: true
     }
   }
   anchor_generator {
     multiscale_anchor_generator {
       min_level: 3
       max_level: 7
       anchor_scale: 4.0
       aspect_ratios: 1.0
       aspect_ratios: 2.0
       aspect_ratios: 0.5
       scales_per_octave: 2
     }
   }
   post_processing {
     batch_non_max_suppression {
       score_threshold: 0.0099999997764825529e-09
       iou_threshold: 0.6000000238418579
       max_detections_per_class: 100
       max_total_detections: 100
       use_static_shapes: false
     }
     score_converter: SIGMOID
   }
   normalize_loss_by_num_matches: true
   loss {
     localization_loss {
       weighted_smooth_l1 {
       }
     }
     classification_loss {
       weighted_sigmoid_focal {
         gamma: 2.0
         alpha: 0.25
       }
     }
     classification_weight: 1.0
     localization_weight: 1.0
   }
   encode_background_as_zeros: true
   normalize_loc_loss_by_codesize: true
   inplace_batchnorm_update: true
   freeze_batchnorm: false
 },
 'train_config': batch_size: 4
 data_augmentation_options {
   random_horizontal_flip {
   }
 }
 data_augmentation_options {
   random_crop_image {
     min_object_covered: 0.0
     min_aspect_ratio: 0.75
     max_aspect_ratio: 3.0
     min_area: 0.75
     max_area: 1.0
     overlap_thresh: 0.0
   }
 }
 sync_replicas: true
 optimizer {
   momentum_optimizer {
     learning_rate {
       cosine_decay_learning_rate {
         learning_rate_base: 0.07999999211488066
         total_steps: 50000
         warmup_learning_rate: 0.026666000485420227
         warmup_steps: 1000
       }
     }
     momentum_optimizer_value: 0.8999999761581421
   }
   use_moving_average: false
 }
 fine_tune_checkpoint: "Tensorflow\workspace/pre-trained-models/ssd_mobilenet_v2_fpnlite_320x320_coco17_tpu-8/checkpoint\ckpt-0"
 fine_tune_checkpoint_type: "detection"
 num_steps: 50000
 startup_delay_steps: 0.0
 replicas_to_aggregate: 8
 max_number_of_boxes: 100
 unpad_groundtruth_tensors: false
 fine_tune_checkpoint_type: "detection"
 fine_tune_checkpoint_version: V2
 'train_input_config': label_map_path: "Tensorflow\workspace/annotations/label_map.pbtxt"
 tf_record_input_reader {
   input_path: "Tensorflow\workspace/annotations/train.record"
 }
 'eval_config': metrics_set: "coco_detection_metrics"
 use_moving_averages: false
 'eval_input_config': label_map_path: "Tensorflow\workspace/annotations/label_map.pbtxt"
 shuffle: false
 num_epochs: 1
 tf_record_input_reader {
   input_path: "Tensorflow\workspace/annotations/test.record"
 }
 'eval_input_configs': label_map_path: "Tensorflow\workspace/annotations/label_map.pbtxt"
 shuffle: false
 num_epochs: 1
 tf_record_input_reader {
   input_path: "Tensorflow\workspace/annotations/test.record"
 }}
```

```
pipeline_config = pipeline_pb2.TrainEvalPipelineConfig()
with tf.io.gfile.GFile(CONFIG_PATH, "r") as f:
    proto_str = f.read()
    text_format.Merge(proto_str, pipeline_config)
```

```
pipeline_config.model.ssd.num_classes = 2
pipeline_config.train_config.batch_size = 4
pipeline_config.train_config.fine_tune_checkpoint = PRETRAINED_MODEL_PATH+'/ssd_mobilenet_v2_fpnlite_320x320_coco17_tpu-8/checkpoint\ckpt-0'
pipeline_config.train_config.fine_tune_checkpoint_type = "detection"
pipeline_config.train_input_reader.label_map_path= ANNOTATION_PATH + '/label_map.pbtxt'
pipeline_config.train_input_reader.tf_record_input_reader.input_path[:] = [ANNOTATION_PATH + '/train.record']
pipeline_config.eval_input_reader[0].label_map_path = ANNOTATION_PATH + '/label_map.pbtxt'
pipeline_config.eval_input_reader[0].tf_record_input_reader.input_path[:] = [ANNOTATION_PATH + '/test.record']
```

```
config_text = text_format.MessageToString(pipeline_config)
with tf.io.gfile.GFile(CONFIG_PATH, "wb") as f:
    f.write(config_text)
```

Train Model

```
print("""python {}/research/object_detection/model_main_tf2.py --model_dir={}/{} --pipeline_config_path={}/{}/pipeline.config --num_train_steps=5000""".format(APIMODEL_PATH, MODEL_PATH, CUSTOM_MODEL_NAME, MODEL_PATH, CUSTOM_MODEL_NAME))
```

```
python Tensorflow/models/research/object_detection/model_main_tf2.py --model_dir=Tensorflow\workspace/models/my_ssd_mobnet --pipeline_config_path=Tensorflow\workspace/models/my_ssd_mobnet/pipeline.config --num_train_steps=5000
```

```
import os
from object_detection.utils import label_map_util
from object_detection.utils import visualization_utils as viz_utils
from object_detection.builders import model_builder
```

```
# Load pipeline config and build a detection model
configs = config_util.get_configs_from_pipeline_file(CONFIG_PATH)
detection_model = model_builder.build(model_config=configs['model'], is_training=False)

# Restore checkpoint
ckpt = tf.compat.v2.train.Checkpoint(model=detection_model)
ckpt.restore(os.path.join(CHECKPOINT_PATH, 'ckpt-0')).expect_partial()

@tf.function
def detect_fn(image):
    image, shapes = detection_model.preprocess(image)
    prediction_dict = detection_model.predict(image, shapes)
    detections = detection_model.postprocess(prediction_dict, shapes)
    return detections
```

Detect in Real Time

```
import cv2
import numpy as np
```

```
category_index = label_map_util.create_category_index_from_labelmap(ANNOTATION_PATH+'/label_map.pbtxt')
```

```
# Setup capture

cap = cv2.VideoCapture(0)
width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))

while True:
    ret, frame = cap.read()
    image_np = np.array(frame)

    input_tensor = tf.convert_to_tensor(np.expand_dims(image_np, 0), dtype=tf.float32)
    detections = detect_fn(input_tensor)

    num_detections = int(detections.pop('num_detections'))
    detections = {key: value[0, :num_detections].numpy()
                  for key, value in detections.items()}
    detections['num_detections'] = num_detections

    # detection_classes should be ints.
    detections['detection_classes'] = detections['detection_classes'].astype(np.int64)

    label_id_offset = 1
    image_np_with_detections = image_np.copy()

    viz_utils.visualize_boxes_and_labels_on_image_array(
                image_np_with_detections,
                detections['detection_boxes'],
                detections['detection_classes']+label_id_offset,
                detections['detection_scores'],
                category_index,
                use_normalized_coordinates=True,
                max_boxes_to_draw=5,
                min_score_thresh=.8,
                agnostic_mode=False)

    cv2.imshow('object detection',  cv2.resize(image_np_with_detections, (800, 600)))

    if cv2.waitKey(1) & 0xFF == ord('q'):
        cap.release()
        break
```

```
detections = detect_fn(input_tensor)
```

```
from matplotlib import pyplot as plt
```

Thank You...