Object Detection API Demo

Welcome to the Object Detection API. This notebook will walk you step by step through the process of using a pre-trained model to detect objects in an image.

Imports

In [1]:
```
import numpy as np
import random
import os
import glob
import six.moves.urllib as urllib
import sys
import tarfile
import tensorflow as tf
import zipfile
import pathlib

from collections import defaultdict
from io import StringIO
from matplotlib import pyplot as plt
from PIL import Image
from IPython.display import display

import matplotlib.pyplot as plt

from object_detection.protos import string_int_label_map_pb2
from google.protobuf import text_format
```

Import the object detection module.

In [2]:
```
from object_detection.utils import ops as utils_ops
from object_detection.utils import label_map_util
from object_detection.utils import visualization_utils as vis_util
```

Patches:

In [3]:
```
# patch tf1 into `utils.ops`
utils_ops.tf = tf.compat.v1
```

In [4]:
```
# Load and check the dataset type
dataset_path = "workspace/training_demo/images"

train_images = glob.glob(os.path.join(dataset_path, "train", "*.jpg"))
train_labels = glob.glob(os.path.join(dataset_path, "train", "*.xml"))

val_images = glob.glob(os.path.join(dataset_path, "valid", "*.jpg"))
val_labels = glob.glob(os.path.join(dataset_path, "valid", "*.xml"))

test_images = glob.glob(os.path.join(dataset_path, "test", "*.jpg"))
test_labels = glob.glob(os.path.join(dataset_path, "test", "*.xml"))

print(f"Train: {len(train_images)} images, {len(train_labels)} XMLs")
print(f"Val: {len(val_images)} images, {len(val_labels)} XMLs")
print(f"Test: {len(test_images)} images, {len(test_labels)} XMLs")
```

```
Train: 150 images, 150 XMLs
Val: 20 images, 20 XMLs
Test: 30 images, 30 XMLs
```

Model preparation

Variables

Any model exported using the `export_inference_graph.py` tool can be loaded here simply by changing the path.

By default we use an "SSD with Mobilenet" model here. See the detection model zoo for a list of other models that can be run out-of-the-box with varying speeds and accuracies.

Loader

In [5]:
```
# Load the trained model
MODEL_DIR = "workspace/training_demo/exported-models/my_model/saved_model"
detection_model = tf.saved_model.load(MODEL_DIR)
```

Loading label map

Label maps map indices to category names, so that when our convolution network predicts `5`, we know that this corresponds to `airplane`. Here we use internal utility functions, but anything that returns a dictionary mapping integers to appropriate string labels would be fine

In [6]:
```
# Load label map
PATH_TO_LABELS = "workspace/training_demo/annotations/label_map.pbtxt"
category_index = label_map_util.create_category_index_from_labelmap(PATH_TO_LABELS, use_display_name=True)
```

In [7]:
```
# Get a list of test images
TEST_IMAGE_DIR = "workspace/training_demo/images/test/"
image_files = sorted([
    os.path.join(TEST_IMAGE_DIR, f)
    for f in os.listdir(TEST_IMAGE_DIR) if f.endswith(('.jpg', '.png', '.jpeg'))
])

print("The first 5 images:", image_files[:5])
```

```
The first 5 images: ['workspace/training_demo/images/test/img-152-_jpg.rf.0e7e3d4d5b37e1979c01608199cf69c7.jpg', 'workspace/training_demo/images/test/img-153-_jpg.rf.5a393ffdda6c0820b2c23ebb48aba7b1.jpg', 'workspace/training_demo/images/test/img-155-_jpg.rf.d7966aef18b8bc85a46f6e690104b172.jpg', 'workspace/training_demo/images/test/img-156-_jpg.rf.2e29dfbafec53f3dffd2801359a764ed.jpg', 'workspace/training_demo/images/test/img-159-_jpg.rf.1d1293bd999d94f429d9b15bc015428e.jpg']
```

In [8]:
```
# If you want to test the code with your images, just add path to the images to the TEST_IMAGE_PATHS.
IMAGE_PATH = "workspace/training_demo/images/test/"
```

Detection

Load an object detection model:

In [9]:
```
detection_model = tf.saved_model.load('workspace/training_demo/exported-models/my_model/saved_model')
```

Check the model's input signature, it expects a batch of 3-color images of type uint8:

In [10]:
```
# Check the Input of the model
input_tensors = detection_model.signatures['serving_default'].inputs
filtered_inputs = [t for t in input_tensors if "unknown" not in t.name]
print("Inout model:", filtered_inputs)
```

```
Inout model: [<tf.Tensor 'input_tensor:0' shape=(1, None, None, 3) dtype=uint8>]
```

In [11]:
```
# Check the output of model
output_dtypes = detection_model.signatures['serving_default'].output_dtypes
output_shapes = detection_model.signatures['serving_default'].output_shapes
print("Output Dtypes:", {key: value for key, value in output_dtypes.items() if "raw" not in key})
print("Output Shapes:", {key: value for key, value in output_shapes.items() if "raw" not in key})
```

```
Output Dtypes: {'num_detections': tf.float32, 'detection_multiclass_scores': tf.float32, 'detection_scores': tf.float32, 'detec
tion_anchor_indices': tf.float32, 'detection_classes': tf.float32, 'detection_boxes': tf.float32}
Output Shapes: {'num_detections': TensorShape([1]), 'detection_multiclass_scores': TensorShape([1, 300, 2]), 'detection_score
s': TensorShape([1, 300]), 'detection_anchor_indices': TensorShape([1, 300]), 'detection_classes': TensorShape([1, 300]), 'dete
ction_boxes': TensorShape([1, 300, 4])}
```

And returns several outputs:

In [12]:
```
detection_model.signatures['serving_default'].output_dtypes
```

Out[12]:
```
{'raw_detection_scores': tf.float32,
 'num_detections': tf.float32,
 'detection_multiclass_scores': tf.float32,
 'detection_scores': tf.float32,
 'detection_anchor_indices': tf.float32,
 'detection_classes': tf.float32,
 'raw_detection_boxes': tf.float32,
 'detection_boxes': tf.float32}
```

In [13]:
```
detection_model.signatures['serving_default'].output_shapes
```

Out[13]:
```
{'raw_detection_scores': TensorShape([1, 300, 2]),
 'num_detections': TensorShape([1]),
 'detection_multiclass_scores': TensorShape([1, 300, 2]),
 'detection_scores': TensorShape([1, 300]),
 'detection_anchor_indices': TensorShape([1, 300]),
 'detection_classes': TensorShape([1, 300]),
 'raw_detection_boxes': TensorShape([1, 300, 4]),
 'detection_boxes': TensorShape([1, 300, 4])}
```

Add a wrapper function to call the model, and cleanup the outputs:

In [14]:
```python
def run_inference_for_single_image(model, image):
    # Run inference on an image, return prediction results
    image = np.asarray(image)  # Convert images to numpy array
    input_tensor = tf.convert_to_tensor(image)[tf.newaxis, ...]  # Add batch dimension

    # Run inference with model
    model_fn = model.signatures['serving_default']
    output_dict = model_fn(input_tensor)

    # Process the output results
    num_detections = int(output_dict.pop('num_detections'))
    output_dict = {key: value[0, :num_detections].numpy() for key, value in output_dict.items()}
    output_dict['num_detections'] = num_detections
    output_dict['detection_classes'] = output_dict['detection_classes'].astype(np.int64)

    # Handle models with masks
    if 'detection_masks' in output_dict:
        detection_masks_reframed = utils_ops.reframe_box_masks_to_image_masks(
            output_dict['detection_masks'], output_dict['detection_boxes'],
            image.shape[0], image.shape[1])
        detection_masks_reframed = tf.cast(detection_masks_reframed > 0.5, tf.uint8)
        output_dict['detection_masks_reframed'] = detection_masks_reframed.numpy()

    return output_dict
```

Run it on each test image and show the results:

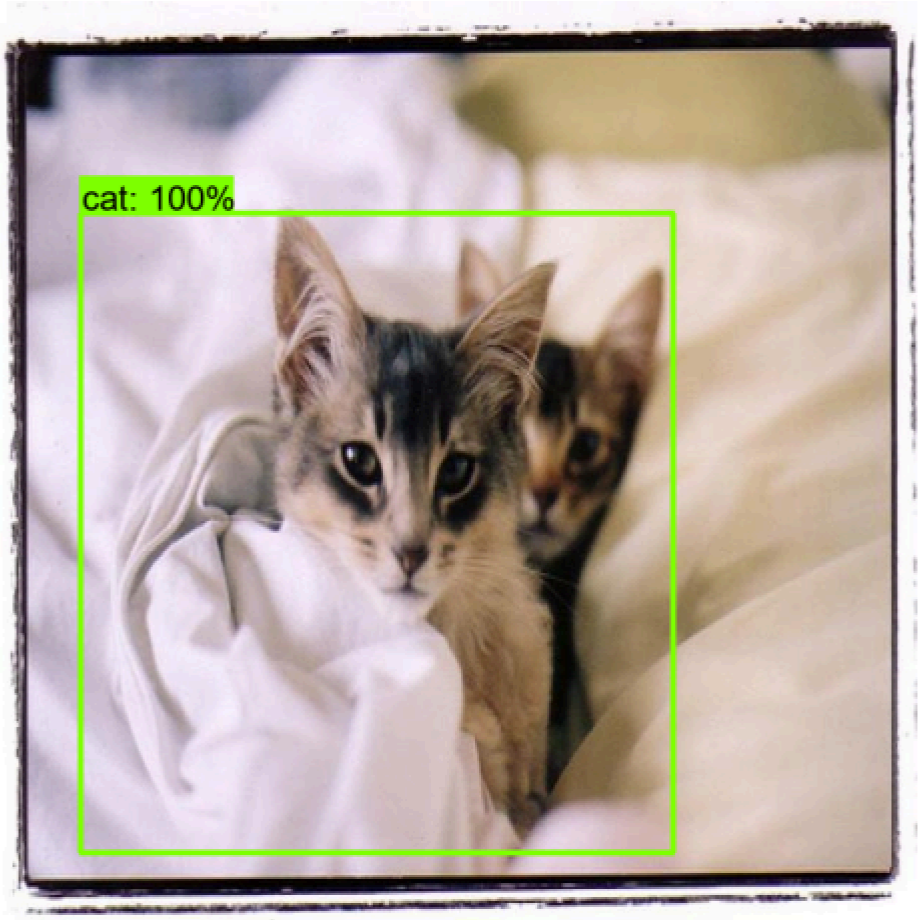
In [15]:
```
def show_inference(model, image_path):
    image_np = np.array(Image.open(image_path))
    output_dict = run_inference_for_single_image(model, image_np)  # Run the model

    # Draw bounding boxes on images
    vis_util.visualize_boxes_and_labels_on_image_array(
        image_np,
        output_dict['detection_boxes'],
        output_dict['detection_classes'],
        output_dict['detection_scores'],
        category_index,
        instance_masks=output_dict.get('detection_masks_reframed', None),
        use_normalized_coordinates=True,
        line_thickness=4)

    # Show the images
    plt.figure(figsize=(10, 6))
    plt.imshow(image_np)
    plt.axis("off")
    plt.show()
    plt.close()
```

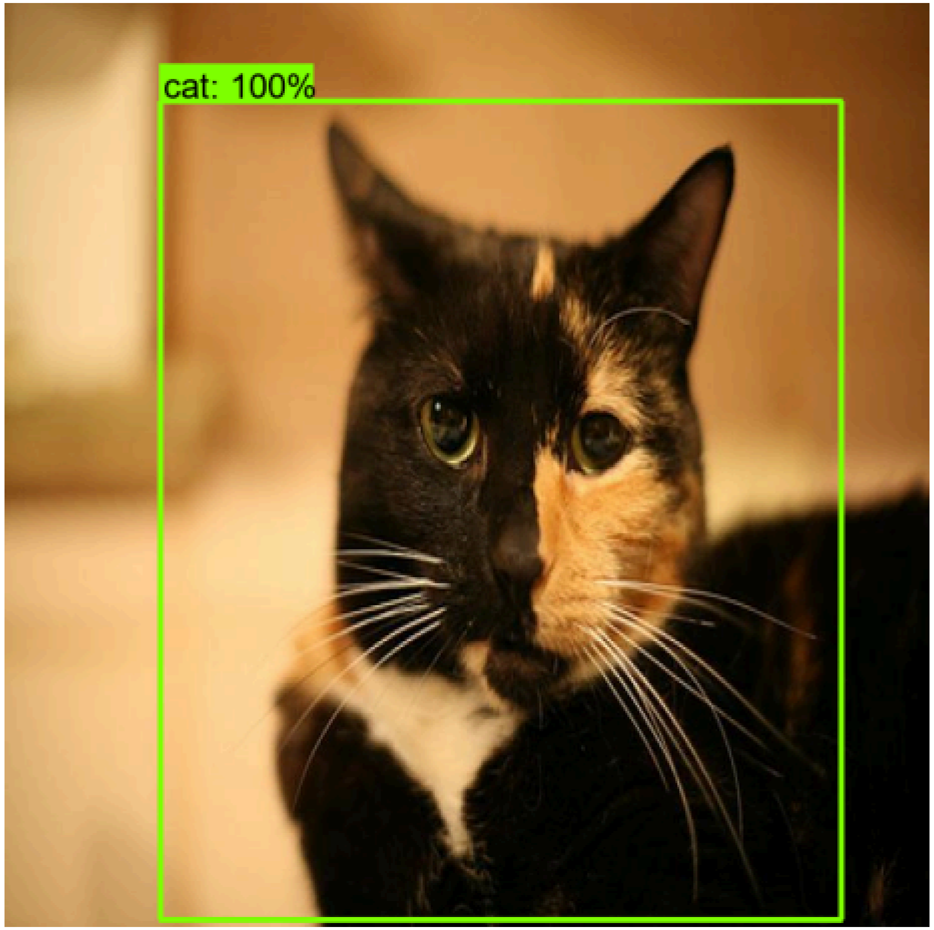
In [16]:
```
%matplotlib inline
# Randomly select 5 images from the list
random_images = random.sample(image_files, 5)

# Run inference on 5 randomly selected images
for img_path in random_images:
    show_inference(detection_model, img_path)
```


cat: 100%

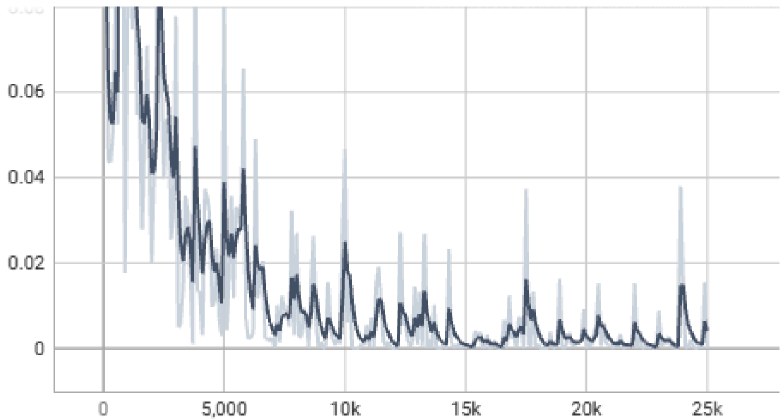

cat: 100%


cat: 100%

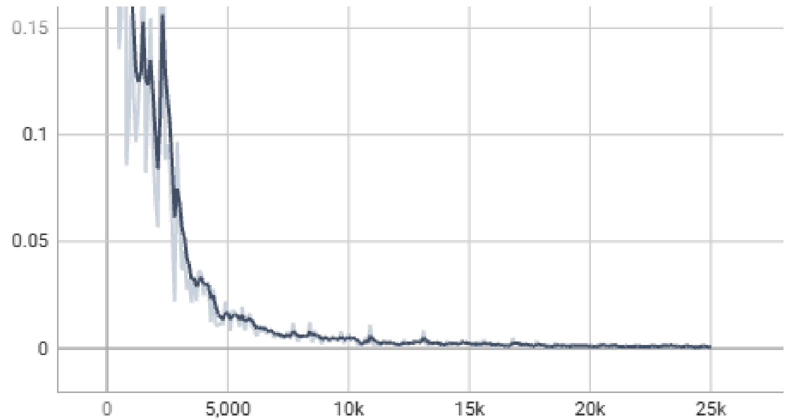

cat: 100%


cat: 100%

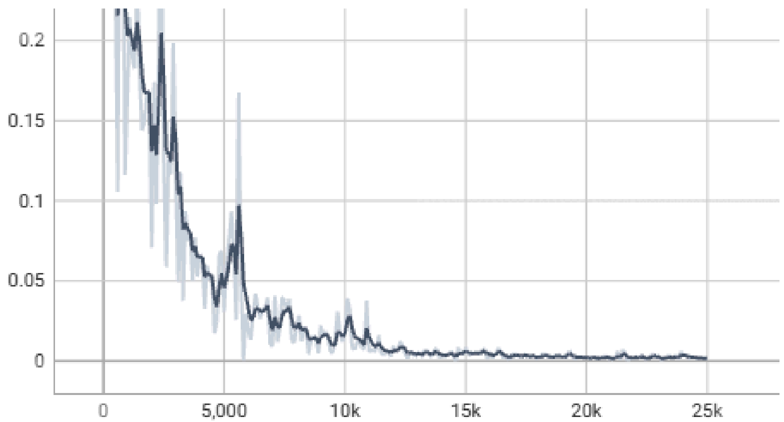
Loss/BoxClassifierLoss/classification_loss



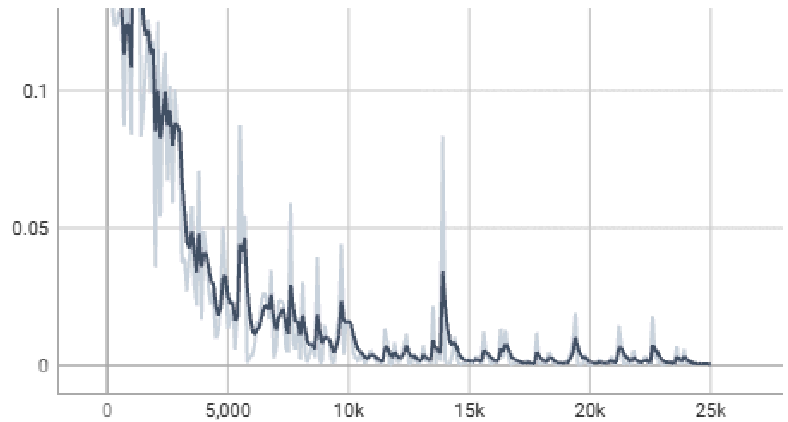
Loss/BoxClassifierLoss/localization_loss



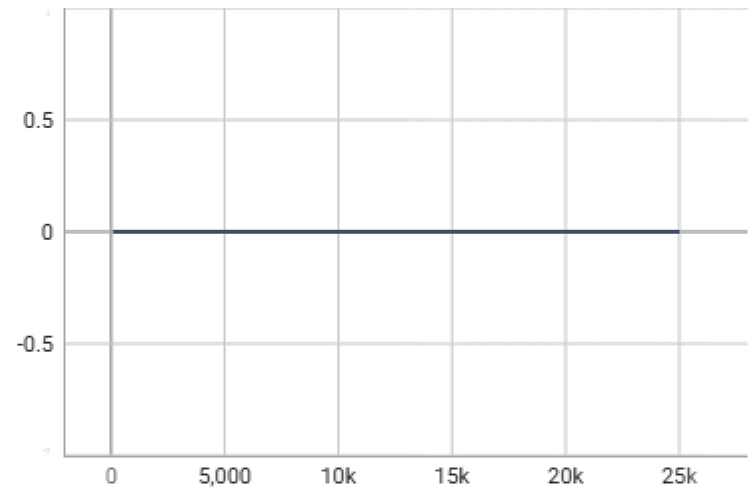
Loss/RPNLoss/localization_loss



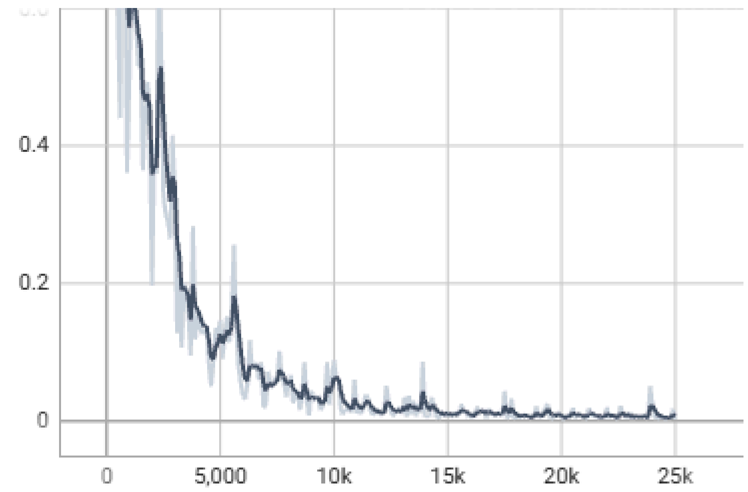
Loss/RPNLoss/objectness_loss



Loss/regularization_loss



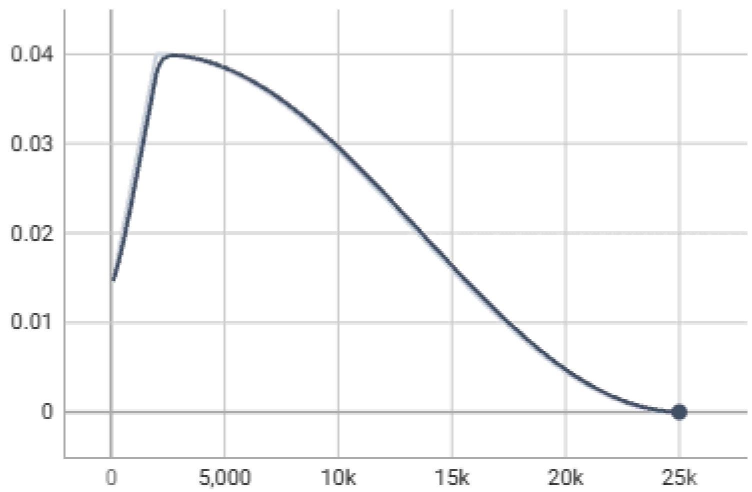
Loss/total_loss



learning_rate

Run	Smoothed	Value	Step	Time	Relative
train	1.3e-5	0	25,000	3/2/25, 3:35 AM	1.596 hr

learning_rate



TensorBoard:

- Train the model on the cat dataset (~25,000 steps).
- Classification Loss, Localization Loss, Objectness Loss all gradually decrease → Good learning model.
- Learning Rate gradually decreases → Stable optimization process.
- Training time: 1 hour 36 minutes

Conclusion:

Dataset:

- The dataset includes 200 cat images downloaded from Google (150 train, 20 validation, 30 test).
- Label the image with Roboflow and save in Pascal VOC .xml format
- Convert to TFRecord for use with TensorFlow Object Detection API.

Model:

- Using Faster R-CNN model with ResNet101 640 x 640 trained on COCO dataset.
- Fine-tune the model on the dataset by changing the .config config.

Result:

- Evaluate the model with the validation set (20 images).
- Tested on the test set (30 images).
- The model works well, accurately detecting cats in photos.