Week 2 Assignment: Zombie Detection

Welcome to this week's programming assignment! You will use the Object Detection API and retrain RetinaNet to spot Zombies using just 5 training images. You will setup the model to restore pretrained weights and fine tune the classification layers.

Important: *This colab notebook has read-only access so you won't be able to save your changes. If you want to save your work periodically, please click* `File -> Save a Copy in Drive` *to create a copy in your account, then work from there.*

zombie: 100.0%

Exercises

Installation

You'll start by installing the Tensorflow 2 Object Detection API.

```
!pip install -U --pre tensorflow=="2.2.0"
```

```
     Collecting tensorflow==2.2.0
       Downloading https://files.pythonhosted.org/packages/4c/1a/0d79814736cfecc825ab8094b39648cc9c
          |████████████████████████████████| 516.2MB 31kB/s
     Requirement already satisfied, skipping upgrade: wrapt>=1.11.1 in /usr/local/lib/python3.7/dis
     Collecting tensorboard<2.3.0,>=2.2.0
       Downloading https://files.pythonhosted.org/packages/1d/74/0a6fcb206dcc72a6da9a62dd81784bfdbf
          |████████████████████████████████| 3.0MB 69kB/s
     Requirement already satisfied, skipping upgrade: opt-einsum>=2.3.2 in /usr/local/lib/python3.7
     Requirement already satisfied, skipping upgrade: six>=1.12.0 in /usr/local/lib/python3.7/dist-
     Requirement already satisfied, skipping upgrade: absl-py>=0.7.0 in /usr/local/lib/python3.7/di
     Requirement already satisfied, skipping upgrade: grpcio>=1.8.6 in /usr/local/lib/python3.7/dis
     Requirement already satisfied, skipping upgrade: astunparse==1.6.3 in /usr/local/lib/python3.7
     Collecting tensorflow-estimator<2.3.0,>=2.2.0
       Downloading https://files.pythonhosted.org/packages/a4/f5/926ae53d6a226ec0fda5208e0e581cffed
          |████████████████████████████████| 460kB 49.6MB/s
     Collecting h5py<2.11.0,>=2.10.0
       Downloading https://files.pythonhosted.org/packages/3f/c0/abde58b837e066bca19a3f7332d9d04935
          |████████████████████████████████| 2.9MB 41.4MB/s
     Requirement already satisfied, skipping upgrade: scipy==1.4.1; python_version >= "3" in /usr/l
     Requirement already satisfied, skipping upgrade: protobuf>=3.8.0 in /usr/local/lib/python3.7/d
     Requirement already satisfied, skipping upgrade: wheel>=0.26; python_version >= "3" in /usr/lo
     Requirement already satisfied, skipping upgrade: google-pasta>=0.1.8 in /usr/local/lib/python3
     Collecting gast==0.3.3
       Downloading https://files.pythonhosted.org/packages/d6/84/759f5dd23fec8ba71952d97bcc7e2c9d7d
     Requirement already satisfied, skipping upgrade: keras-preprocessing>=1.1.0 in /usr/local/lib/
     Requirement already satisfied, skipping upgrade: termcolor>=1.1.0 in /usr/local/lib/python3.7/
     Requirement already satisfied, skipping upgrade: numpy<2.0,>=1.16.0 in /usr/local/lib/python3.
     Requirement already satisfied, skipping upgrade: google-auth-oauthlib<0.5,>=0.4.1 in /usr/loca
     Requirement already satisfied, skipping upgrade: tensorboard-plugin-wit>=1.6.0 in /usr/local/l
     Requirement already satisfied, skipping upgrade: werkzeug>=0.11.15 in /usr/local/lib/python3.7
     Requirement already satisfied, skipping upgrade: google-auth<2,>=1.6.3 in /usr/local/lib/pytho
     Requirement already satisfied, skipping upgrade: requests<3,>=2.21.0 in /usr/local/lib/python3
```

```
# uncomment the next line if you want to delete an existing models directory
!rm -rf ./models/

# clone the Tensorflow Model Garden
!git clone --depth 1 https://github.com/tensorflow/models/
```

```
     Cloning into 'models'...
     remote: Enumerating objects: 2650, done.
     remote: Counting objects: 100% (2650/2650), done.
     remote: Compressing objects: 100% (2202/2202), done.
     remote: Total 2650 (delta 671), reused 1299 (delta 416), pack-reused 0
     Receiving objects: 100% (2650/2650), 32.62 MiB | 33.37 MiB/s, done.
     Resolving deltas: 100% (671/671), done.

     ERROR: tf-models-official 2.5.0 has requirement tensorflow>=2.5.0, but you'll have tensorflow
```

```
# install the Object Detection API
!cd models/research/ && protoc object_detection/protos/*.proto --python_out=. && cp object_detection
```

```
     Requirement already satisfied: pyasn1>=0.1.7 in /usr/local/lib/python3.7/dist-packages (fro
     Requirement already satisfied: pyasn1-modules>=0.0.5 in /usr/local/lib/python3.7/dist-packa
     Requirement already satisfied: rsa>=3.1.4 in /usr/local/lib/python3.7/dist-packages (from o
     Requirement already satisfied: docopt in /usr/local/lib/python3.7/dist-packages (from hdfs<
     Requirement already satisfied: termcolor in /usr/local/lib/python3.7/dist-packages (from te
     Requirement already satisfied: tensorflow-metadata in /usr/local/lib/python3.7/dist-package
     Requirement already satisfied: dm-tree in /usr/local/lib/python3.7/dist-packages (from tens
     Requirement already satisfied: promise in /usr/local/lib/python3.7/dist-packages (from tens
     Requirement already satisfied: tqdm in /usr/local/lib/python3.7/dist-packages (from tensorf
     Requirement already satisfied: attrs>=18.1.0 in /usr/local/lib/python3.7/dist-packages (fro
     Requirement already satisfied: importlib-resources; python_version < "3.9" in /usr/local/li
     Requirement already satisfied: portalocker==2.0.0 in /usr/local/lib/python3.7/dist-packages
     Requirement already satisfied: h5py~=3.1.0 in /usr/local/lib/python3.7/dist-packages (from
     Requirement already satisfied: wrapt~=1.12.1 in /usr/local/lib/python3.7/dist-packages (fro
     Requirement already satisfied: wheel~=0.35 in /usr/local/lib/python3.7/dist-packages (from
     Requirement already satisfied: gast==0.4.0 in /usr/local/lib/python3.7/dist-packages (from
     Requirement already satisfied: keras-preprocessing~=1.1.2 in /usr/local/lib/python3.7/dist-
     Requirement already satisfied: tensorflow-estimator<2.6.0,>=2.5.0rc0 in /usr/local/lib/pyth
     Requirement already satisfied: astunparse~=1.6.3 in /usr/local/lib/python3.7/dist-packages
     Requirement already satisfied: opt-einsum~=3.3.0 in /usr/local/lib/python3.7/dist-packages
     Requirement already satisfied: flatbuffers~=1.12.0 in /usr/local/lib/python3.7/dist-package
     Requirement already satisfied: tensorboard~=2.5 in /usr/local/lib/python3.7/dist-packages (
     Requirement already satisfied: keras-nightly~=2.5.0.dev in /usr/local/lib/python3.7/dist-pa
     Requirement already satisfied: google-pasta~=0.2 in /usr/local/lib/python3.7/dist-packages
     Requirement already satisfied: google-cloud-core<2.0dev,>=1.0.3 in /usr/local/lib/python3.7
     Requirement already satisfied: google-resumable-media!=0.4.0,<0.5.0dev,>=0.3.1 in /usr/loca
     Requirement already satisfied: google-api-core<2dev,>=1.21.0 in /usr/local/lib/python3.7/di
     Requirement already satisfied: google-auth-httplib2>=0.0.3 in /usr/local/lib/python3.7/dist
     Requirement already satisfied: google-auth>=1.16.0 in /usr/local/lib/python3.7/dist-package
     Requirement already satisfied: uritemplate<4dev,>=3.0.0 in /usr/local/lib/python3.7/dist-pa
     Requirement already satisfied: python-slugify in /usr/local/lib/python3.7/dist-packages (fr
     Requirement already satisfied: scikit-learn>=0.21.3 in /usr/local/lib/python3.7/dist-packag
     Requirement already satisfied: typeguard>=2.7 in /usr/local/lib/python3.7/dist-packages (fr
     Requirement already satisfied: googleapis-common-protos<2,>=1.52.0 in /usr/local/lib/python
     Requirement already satisfied: zipp>=0.4; python_version < "3.8" in /usr/local/lib/python3.
     Requirement already satisfied: cached-property; python_version < "3.8" in /usr/local/lib/py
     Requirement already satisfied: werkzeug>=0.11.15 in /usr/local/lib/python3.7/dist-packages
     Requirement already satisfied: google-auth-oauthlib<0.5,>=0.4.1 in /usr/local/lib/python3.7
     Requirement already satisfied: markdown>=2.6.8 in /usr/local/lib/python3.7/dist-packages (f
     Requirement already satisfied: tensorboard-data-server<0.7.0,>=0.6.0 in /usr/local/lib/pyth
     Requirement already satisfied: tensorboard-plugin-wit>=1.6.0 in /usr/local/lib/python3.7/di
     Requirement already satisfied: packaging>=14.3 in /usr/local/lib/python3.7/dist-packages (f
     Requirement already satisfied: cachetools<5.0,>=2.0.0 in /usr/local/lib/python3.7/dist-pack
     Requirement already satisfied: text-unidecode>=1.3 in /usr/local/lib/python3.7/dist-package
     Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.7/dist-packages (from
     Requirement already satisfied: requests-oauthlib>=0.7.0 in /usr/local/lib/python3.7/dist-pa
     Requirement already satisfied: importlib-metadata; python_version < "3.8" in /usr/local/lib
     Requirement already satisfied: oauthlib>=3.0.0 in /usr/local/lib/python3.7/dist-packages (f
     Building wheels for collected packages: object-detection
       Building wheel for object-detection (setup.py) ... done
       Created wheel for object-detection: filename=object_detection-0.1-cp37-none-any.whl size=
       Stored in directory: /tmp/pip-ephem-wheel-cache-nt1jp32u/wheels/94/49/4b/39b051683087a22e
     Successfully built object-detection
     Installing collected packages: object-detection
       Found existing installation: object-detection 0.1
         Uninstalling object-detection-0.1:
           Successfully uninstalled object-detection-0.1
     Successfully installed object-detection-0.1
```

Imports

Let's now import the packages you will use in this assignment.

```
import matplotlib
import matplotlib.pyplot as plt

import os
import random
import zipfile
import io
import scipy.misc
import numpy as np

import glob
import imageio
import imageio
from six import BytesIO
from PIL import Image, ImageDraw, ImageFont
from IPython.display import display, Javascript
from IPython.display import Image as IPyImage

try:
  # %tensorflow_version only exists in Colab.
  %tensorflow_version 2.x
except Exception:
  pass

import tensorflow as tf
tf.get_logger().setLevel('ERROR')
```

Exercise 1: Import Object Detection API packages

Import the necessary modules from the `object_detection` package.

- From the utils package:
  - label_map_util
  - config_util: You'll use this to read model configurations from a .config file and then modify that configuration
  - visualization_utils: please give this the alias `viz_utils`, as this is what will be used in some visualization code that is given to you later.
  - colab_utils
- From the builders package:
  - model_builder: This builds your model according to the model configuration that you'll specify.

```
### START CODE HERE (Replace Instances of `None` with your code) ###
# import the label map utility module
from object_detection.utils import label_map_util

# import module for reading and updating configuration files.
from object_detection.utils import config_util

# import module for visualization. use the alias `viz_utils`
from object_detection.utils import visualization_utils as viz_utils

# import module for building the detection model
from object_detection.builders import model_builder
### END CODE HERE ###

# import module for utilities in Colab
from object_detection.utils import colab_utils
```

Utilities

You'll define a couple of utility functions for loading images and plotting detections. This code is provided for you.

```python
def load_image_into_numpy_array(path):
    """Load an image from file into a numpy array.

    Puts image into numpy array to feed into tensorflow graph.
    Note that by convention we put it into a numpy array with shape
    (height, width, channels), where channels=3 for RGB.

    Args:
    path: a file path.

    Returns:
    uint8 numpy array with shape (img_height, img_width, 3)
    """

    img_data = tf.io.gfile.GFile(path, 'rb').read()
    image = Image.open(BytesIO(img_data))
    (im_width, im_height) = image.size

    return np.array(image.getdata()).reshape(
        (im_height, im_width, 3)).astype(np.uint8)


def plot_detections(image_np,
                    boxes,
                    classes,
                    scores,
                    category_index,
                    figsize=(12, 16),
                    image_name=None):
    """Wrapper function to visualize detections.

    Args:
    image_np: uint8 numpy array with shape (img_height, img_width, 3)
    boxes: a numpy array of shape [N, 4]
    classes: a numpy array of shape [N]. Note that class indices are 1-based,
          and match the keys in the label map.
    scores: a numpy array of shape [N] or None.  If scores=None, then
          this function assumes that the boxes to be plotted are groundtruth
          boxes and plot all boxes as black with no classes or scores.
    category_index: a dict containing category dictionaries (each holding
          category index `id` and category name `name`) keyed by category indices.
    figsize: size for the figure.
    image_name: a name for the image file.
    """

    image_np_with_annotations = image_np.copy()

    viz_utils.visualize_boxes_and_labels_on_image_array(
        image_np_with_annotations,
        boxes,
        classes,
        scores,
        category_index,
        use_normalized_coordinates=True,
        min_score_thresh=0.8)

    if image_name:
        plt.imsave(image_name, image_np_with_annotations)
```

```
  else:
      plt.imshow(image_np_with_annotations)
```

Download the Zombie data

Now you will get 5 images of zombies that you will use for training.

- The zombies are hosted in a Google bucket.
- You can download and unzip the images into a local `training/` directory by running the cell below.

```
# uncomment the next 2 lines if you want to delete an existing zip and training directory
# !rm training-zombie.zip
# !rm -rf ./training

# download the images
!wget --no-check-certificate \
    https://storage.googleapis.com/laurencemoroney-blog.appspot.com/training-zombie.zip \
    -O ./training-zombie.zip

# unzip to a local directory
local_zip = './training-zombie.zip'
zip_ref = zipfile.ZipFile(local_zip, 'r')
zip_ref.extractall('./training')
zip_ref.close()
```

```
--2021-05-22 23:10:50--  https://storage.googleapis.com/laurencemoroney-blog.appspot.com/train
Resolving storage.googleapis.com (storage.googleapis.com)... 74.125.137.128, 142.250.141.128,
Connecting to storage.googleapis.com (storage.googleapis.com)|74.125.137.128|:443... connected
HTTP request sent, awaiting response... 200 OK
Length: 1915446 (1.8M) [application/zip]
Saving to: ‘./training-zombie.zip’

./training-zombie.z 100%[===================>]   1.83M  --.-KB/s    in 0.01s

2021-05-22 23:10:50 (176 MB/s) - ‘./training-zombie.zip’ saved [1915446/1915446]
```

Exercise 2: Visualize the training images

Next, you'll want to inspect the images that you just downloaded.

- Please replace instances of `None` below to load and visualize the 5 training images.
- You can inspect the *training* directory (using the `Files` button on the left side of this Colab) to see the filenames of the zombie images. The paths for the images will look like this:

```
./training/training-zombie1.jpg
./training/training-zombie2.jpg
./training/training-zombie3.jpg
./training/training-zombie4.jpg
./training/training-zombie5.jpg
```

- To set file paths, you'll use os.path.join. As an example, if you wanted to create the path './parent_folder/file_name1.txt', you could write:

```
os.path.join('parent_folder', 'file_name', str(1), '.txt')
```

- You should see the 5 training images after running this cell. If not, please inspect your code,

```
%matplotlib inline

### START CODE HERE (Replace Instances of `None` with your code) ###

# assign the name (string) of the directory containing the training images
train_image_dir = './training'

# declare an empty list
train_images_np = []

# run a for loop for each image
for i in range(1, 6):

    # define the path (string) for each image
    image_path = os.path.join(train_image_dir, f'training-zombie{i}.jpg')
    print(image_path)

    # load images into numpy arrays and append to a list
    train_images_np.append(load_image_into_numpy_array(image_path))
### END CODE HERE ###

# configure plot settings via rcParams
plt.rcParams['axes.grid'] = False
plt.rcParams['xtick.labelsize'] = False
plt.rcParams['ytick.labelsize'] = False
plt.rcParams['xtick.top'] = False
plt.rcParams['xtick.bottom'] = False
plt.rcParams['ytick.left'] = False
plt.rcParams['ytick.right'] = False
plt.rcParams['figure.figsize'] = [14, 7]

# plot images
for idx, train_image_np in enumerate(train_images_np):
    plt.subplot(1, 5, idx+1)
    plt.imshow(train_image_np)

plt.show()
```

```
./training/training-zombie1.jpg
./training/training-zombie2.jpg
./training/training-zombie3.jpg
./training/training-zombie4.jpg
./training/training-zombie5.jpg
```

Prepare data for training (Optional)

In this section, you will create your ground truth boxes. You can either draw your own boxes or use a prepopulated list of coordinates that we have provided below.

```
# Define the list of ground truth boxes
gt_boxes = []
```

Option 1: draw your own ground truth boxes

If you want to draw your own, please run the next cell and the following test code. If not, then skip these optional cells.

- Draw a box around the zombie in each image.
- Click the `next image` button to go to the next image
- Click `submit` when it says "All images completed!!".
- Make sure to not make the bounding box too big.
  - If the box is too big, the model might learn the features of the background (e.g. door, road, etc) in determining if there is a zombie or not.
- Include the entire zombie inside the box.
- As an example, scroll to the beginning of this notebook to look at the bounding box around the zombie.

```
# Option 1: draw your own ground truth boxes

# annotate the training images
colab_utils.annotate(train_images_np, box_storage_pointer=gt_boxes)
```

```
gt_boxes
```

```
[array([[0.27685185, 0.42438453, 0.73185185, 0.57444314]]),
 array([[0.30851852, 0.46307151, 0.74018518, 0.6084408 ]]),
 array([[0.41185185, 0.1992966 , 0.93518518, 0.33880422]]),
 array([[0.17518518, 0.62954279, 0.78351852, 0.9073857 ]]),
 array([[0.30018518, 0.14302462, 0.83185185, 0.34232122]])]
```

```
# Option 1: draw your own ground truth boxes
# TEST CODE:
try:
  assert(len(gt_boxes) == 5), "Warning: gt_boxes is empty. Did you click `submit`?"

except AssertionError as e:
  print(e)

# checks if there are boxes for all 5 images
for gt_box in gt_boxes:
    try:
      assert(gt_box is not None), "There are less than 5 sets of box coordinates. " \
                                  "Please re-run the cell above to draw the boxes again.\n" \
                                  "Alternatively, you can run the next cell to load pre-determined '
                                  "ground truth boxes."

    except AssertionError as e:
        print(e)
        break


ref_gt_boxes = [
        np.array([[0.27333333, 0.41500586, 0.74333333, 0.57678781]]),
        np.array([[0.29833333, 0.45955451, 0.75666667, 0.61078546]]),
        np.array([[0.40833333, 0.18288394, 0.945, 0.34818288]]),
        np.array([[0.16166667, 0.61899179, 0.8, 0.91910903]]),
        np.array([[0.28833333, 0.12543962, 0.835, 0.35052755]]),
      ]

for gt_box, ref_gt_box in zip(gt_boxes, ref_gt_boxes):
    try:
      assert(np.allclose(gt_box, ref_gt_box, atol=0.04)), "One of the boxes is too big or too small.
                                                          "Please re-draw and make the box tighter a
    except AssertionError as e:
      print(e)
      break
```

```
Warning: gt_boxes is empty. Did you click `submit`?
```

Option 2: use the given ground truth boxes

You can also use this list if you opt not to draw the boxes yourself.

```
# Option 2: use given ground truth boxes
# set this to `True` if you want to override the boxes you drew
override = False

# bounding boxes for each of the 5 zombies found in each image.
# you can use these instead of drawing the boxes yourself.
ref_gt_boxes = [
        np.array([[0.27333333, 0.41500586, 0.74333333, 0.57678781]]),
        np.array([[0.29833333, 0.45955451, 0.75666667, 0.61078546]]),
        np.array([[0.40833333, 0.18288394, 0.945, 0.34818288]]),
        np.array([[0.16166667, 0.61899179, 0.8, 0.91910903]]),
        np.array([[0.28833333, 0.12543962, 0.835, 0.35052755]]),
      ]

# if gt_boxes is empty, use the reference
if not gt_boxes or override is True:
  gt_boxes = ref_gt_boxes

# if gt_boxes does not contain 5 box coordinates, use the reference
for gt_box in gt_boxes:
    try:
      assert(gt_box is not None)

    except:
      gt_boxes = ref_gt_boxes

      break
```

View your ground truth box coordinates

Whether you chose to draw your own or use the given boxes, please check your list of ground truth box coordinates.

```
# print the coordinates of your ground truth boxes
for gt_box in gt_boxes:
  print(gt_box)
```

```
[[0.27333333 0.41500586 0.74333333 0.57678781]]
[[0.29833333 0.45955451 0.75666667 0.61078546]]
[[0.40833333 0.18288394 0.945      0.34818288]]
[[0.16166667 0.61899179 0.8        0.91910903]]
[[0.28833333 0.12543962 0.835      0.35052755]]
```

Below, we add the class annotations. For simplicity, we assume just a single class, though it should be straightforward to extend this to handle multiple classes. We will also convert everything to the format that

Exercise 3: Define the category index dictionary

You'll need to tell the model which integer class ID to assign to the 'zombie' category, and what 'name' to associate with that integer id.

- zombie_class_id: By convention, class ID integers start numbering from 1,2,3, onward.
  - If there is ever a 'background' class, it could be assigned the integer 0, but in this case, you're just predicting the one zombie class.
  - Since you are just predicting one class (zombie), please assign `1` to the zombie class ID.
- category_index: Please define the `category_index` dictionary, which will have the same structure as this:

```
{human_class_id :
{'id'  : human_class_id,
 'name': 'human_so_far'}
}
```

  - Define `category_index` similar to the example dictionary above, except for zombies.
  - This will be used by the succeeding functions to know the class `id` and `name` of zombie images.
- num_classes: Since you are predicting one class, please assign `1` to the number of classes that the model will predict.
  - This will be used during data preprocessing and again when you configure the model.

```
### START CODE HERE (Replace instances of `None` with your code ###

# Assign the zombie class ID
zombie_class_id = 1

# define a dictionary describing the zombie class
category_index = {zombie_class_id : {'id'  : zombie_class_id,  'name': 'zombie'}}

# Specify the number of classes that the model will predict
num_classes = 1
### END CODE HERE ###


# TEST CODE:

print(category_index[zombie_class_id])
```

```
{'id': 1, 'name': 'zombie'}
```

Expected Output:

```
{'id': 1, 'name': 'zombie'}
```

Data preprocessing

You will now do some data preprocessing so it is formatted properly before it is fed to the model:

- Convert the class labels to one-hot representations
- convert everything (i.e. train images, gt boxes and class labels) to tensors.

This code is provided for you.

```
# The `label_id_offset` here shifts all classes by a certain number of indices;
# we do this here so that the model receives one-hot labels where non-background
# classes start counting at the zeroth index.  This is ordinarily just handled
# automatically in our training binaries, but we need to reproduce it here.

label_id_offset = 1
train_image_tensors = []

# lists containing the one-hot encoded classes and ground truth boxes
gt_classes_one_hot_tensors = []
gt_box_tensors = []

for (train_image_np, gt_box_np) in zip(train_images_np, gt_boxes):

    # convert training image to tensor, add batch dimension, and add to list
    train_image_tensors.append(tf.expand_dims(tf.convert_to_tensor(
        train_image_np, dtype=tf.float32), axis=0))

    # convert numpy array to tensor, then add to list
    gt_box_tensors.append(tf.convert_to_tensor(gt_box_np, dtype=tf.float32))

    # apply offset to to have zero-indexed ground truth classes
    zero_indexed_groundtruth_classes = tf.convert_to_tensor(
        np.ones(shape=[gt_box_np.shape[0]], dtype=np.int32) - label_id_offset)

    # do one-hot encoding to ground truth classes
    gt_classes_one_hot_tensors.append(tf.one_hot(
        zero_indexed_groundtruth_classes, num_classes))

print('Done prepping data.')
```

```
Done prepping data.
```

Visualize the zombies with their ground truth bounding boxes

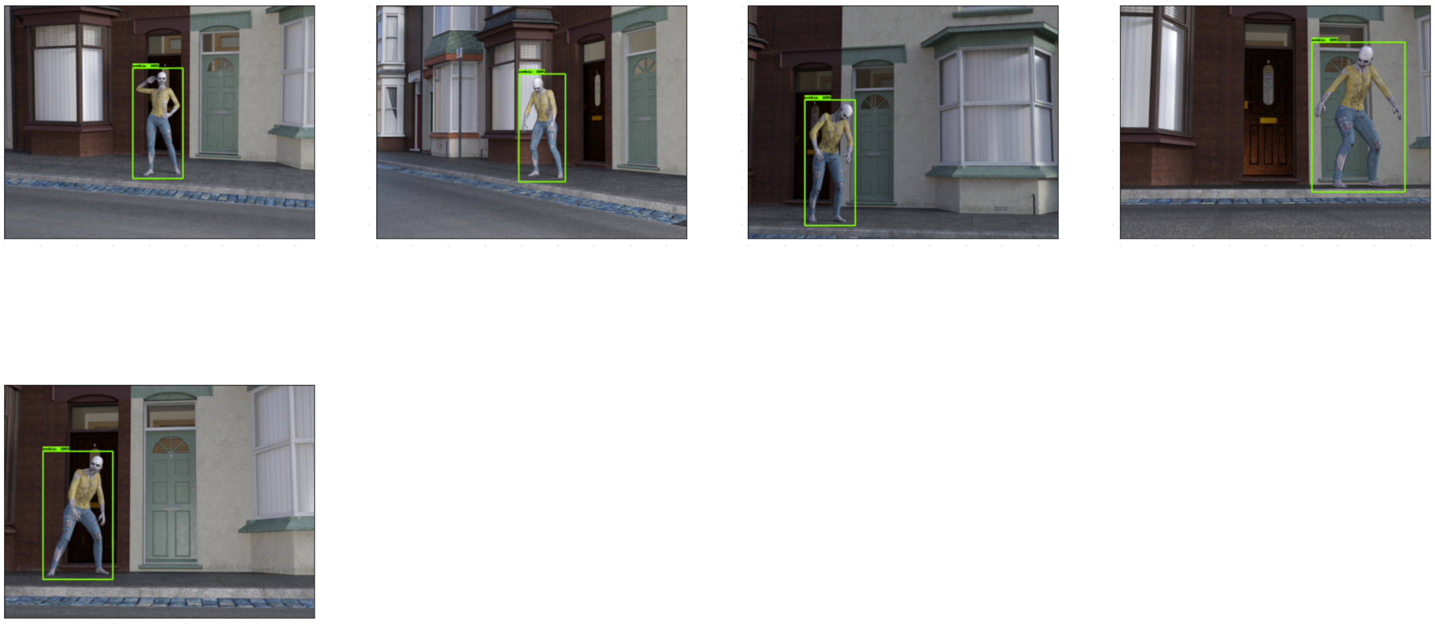
You should see the 5 training images with the bounding boxes after running the cell below. If not, please re-run the annotation tool again or use the prepopulated `gt_boxes` array given.

```
# give boxes a score of 100%
dummy_scores = np.array([1.0], dtype=np.float32)

# define the figure size
plt.figure(figsize=(30, 15))
```

```
# use the `plot_detections()` utility function to draw the ground truth boxes
for idx in range(5):
    plt.subplot(2, 4, idx+1)
    plot_detections(
      train_images_np[idx],
      gt_boxes[idx],
      np.ones(shape=[gt_boxes[idx].shape[0]], dtype=np.int32),
      dummy_scores, category_index)

plt.show()
```



Download the checkpoint containing the pre-trained weights

Next, you will download [RetinaNet](https://github.com/tensorflow/models/tree/master/research/object_detection) and copy it inside the object detection directory.

When working with models that are at the frontiers of research, the models and checkpoints may not yet be organized in a central location like the TensorFlow Garden (https://github.com/tensorflow/models).

- You'll often read a blog post from the researchers, who will usually provide information on:
    - how to use the model
    - where to download the models and pre-trained checkpoints.

It's good practice to do some of this "detective work", so that you'll feel more comfortable when exploring new models yourself! So please try the following steps:

- Go to the TensorFlow Blog, where researchers announce new findings.
- In the search box at the top of the page, search for "retinanet".
- In the search results, click on the blog post titled "TensorFlow 2 meets the Object Detection API" (it may be the first search result).
- Skim through this blog and look for links to either the checkpoints or to Colabs that will show you how to use the checkpoints.
- Try to fill out the following code cell below, which does the following:
  - Download the compressed SSD Resnet 50 version 1, 640 x 640 checkpoint.
  - Untar (decompress) the tar file
  - Move the decompressed checkpoint to `models/research/object_detection/test_data/`

If you want some help getting started, please click on the "Initial Hints" cell to get some hints.

▶ **Initial Hints**

▶ **More Hints**

▼ **Even More Hints**

Even More Hints

- The blog post also links to a notebook titled Eager Few Shot Object Detection Colab
- In this notebook, look for the section titled "Create model and restore weights for all but last layer". The code cell below it shows how to download the exact checkpoint that you're interested in.
- You can also review the lecture videos for this week, which show the same code.

Exercise 4: Download checkpoints

- Download the compressed SSD Resnet 50 version 1, 640 x 640 checkpoint.
- Untar (decompress) the tar file
- Move the decompressed checkpoint to `models/research/object_detection/test_data/`

```
### START CODE HERE ###
# Download the SSD Resnet 50 version 1, 640x640 checkpoint
!wget http://download.tensorflow.org/models/object_detection/tf2/20200711/ssd_resnet50_v1_fpn_640x64

# untar (decompress) the tar file
!tar -xf ssd_resnet50_v1_fpn_640x640_coco17_tpu-8.tar.gz

# copy the checkpoint to the test_data folder models/research/object_detection/test_data/
!mv ssd_resnet50_v1_fpn_640x640_coco17_tpu-8/checkpoint models/research/object_detection/test_data/

### END CODE HERE
```

```
--2021-05-22 23:12:31--  http://download.tensorflow.org/models/object_detection/tf2/20200711/s
Resolving download.tensorflow.org (download.tensorflow.org)... 142.250.141.128, 2607:f8b0:4023
Connecting to download.tensorflow.org (download.tensorflow.org)|142.250.141.128|:80... connect
HTTP request sent, awaiting response... 200 OK
Length: 244817203 (233M) [application/x-tar]
Saving to: 'ssd_resnet50_v1_fpn_640x640_coco17_tpu-8.tar.gz.1'
```

```
ssd_resnet50_v1_fpn 100%[===================>] 233.48M   347MB/s    in 0.7s

2021-05-22 23:12:32 (347 MB/s) - ‘ssd_resnet50_v1_fpn_640x640_coco17_tpu-8.tar.gz.1’ saved [24
```

Configure the model

Here, you will configure the model for this use case.

Exercise 5.1: Locate and read from the configuration file

pipeline_config

- In the Colab, on the left side table of contents, click on the folder icon to display the file browser for the current workspace.
- Navigate to `models/research/object_detection/configs/tf2`. The folder has multiple .config files.
- Look for the file corresponding to ssd resnet 50 version 1 640x640.
- You can double-click the config file to view its contents. This may help you as you complete the next few code cells to configure your model.
- Set the `pipeline_config` to a string that contains the full path to the resnet config file, in other words: `models/research/.../... .config`

configs

If you look at the module config_util that you imported, it contains the following function:

```
def get_configs_from_pipeline_file(pipeline_config_path, config_override=None):
```

- Please use this function to load the configuration from your `pipeline_config`.
    - `configs` will now contain a dictionary.

```
tf.keras.backend.clear_session()


### START CODE HERE ###
# define the path to the .config file for ssd resnet 50 v1 640x640
pipeline_config = 'models/research/object_detection/configs/tf2/ssd_resnet50_v1_fpn_640x640_coco17_t

# Load the configuration file into a dictionary
configs = config_util.get_configs_from_pipeline_file(pipeline_config)

### END CODE HERE ###
# See what configs looks like
configs
```

```
              regularizer {
                l2_regularizer {
                  weight: 0.00039999998989515007
                }
              }
              initializer {
                truncated_normal_initializer {
                  mean: 0.0
                  stddev: 0.029999999329447746
```

```
            stddev: 0.029999999329447746
          }
        }
        activation: RELU_6
        batch_norm {
          decay: 0.996999979019165
          scale: true
          epsilon: 0.0010000000474974513
        }
      }
      override_base_feature_extractor_hyperparams: true
      fpn {
        min_level: 3
        max_level: 7
      }
    }
    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: 0.00039999998989515007
            }
          }
          initializer {
            random_normal_initializer {
              mean: 0.0
              stddev: 0.009999999776482582
            }
          }
          activation: RELU_6
```

Exercise 5.2: Get the model configuration

model_config

- From the `configs` dictionary, access the object associated with the key 'model'.
- `model_config` now contains an object of type `object_detection.protos.model_pb2.DetectionModel`.
- If you print `model_config`, you'll see something like this:

```
  ssd {
    num_classes: 90
    image_resizer {
      fixed_shape_resizer {
        height: 640
        width: 640
      }
    }
    feature_extractor {
  ...
  ...
    freeze_batchnorm: false
```

```
### START CODE HERE ###
# Read in the object stored at the key 'model' of the configs dictionary
model_config = configs['model']

### END CODE HERE
# see what model_config looks like
model_config
```

```
            initializer {
              random_normal_initializer {
                mean: 0.0
                stddev: 0.009999999776482582
              }
            }
            activation: RELU_6
            batch_norm {
              decay: 0.996999979019165
              scale: true
              epsilon: 0.0010000000474974513
            }
          }
          depth: 256
          num_layers_before_predictor: 4
          kernel_size: 3
          class_prediction_bias_init: -4.599999904632568
        }
      }
    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: 9.99999993922529e-09
        iou_threshold: 0.6000000238418579
        max_detections_per_class: 100
        max_total_detections: 100
      }
      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
  }
```

Exercise 5.3: Modify model_config

- Modify num_classes from the default `90` to the `num_classes` that you set earlier in this notebook.
  - num_classes is nested under ssd. You'll need to use dot notation 'obj.x' and NOT bracket notation obj['x']` to access num_classes.
- Freeze batch normalization
  - Batch normalization is not frozen in the default configuration.
  - If you inspect the `model_config` object, you'll see that `freeze_batchnorm` is nested under `ssd` just like `num_classes`.
  - Freeze batch normalization by setting the relevant field to `True`.

```
### START CODE HERE ###
# Modify the number of classes from its default of 90
model_config.ssd.num_classes = num_classes

# Freeze batch normalization
model_config.ssd.freeze_batchnorm = True

### END CODE HERE

# See what model_config now looks like after you've customized it!
model_config
```

```
          }
          initializer {
            random_normal_initializer {
              mean: 0.0
              stddev: 0.009999999776482582
            }
          }
          activation: RELU_6
          batch_norm {
            decay: 0.996999979019165
            scale: true
            epsilon: 0.0010000000474974513
          }
        }
        depth: 256
        num_layers_before_predictor: 4
        kernel_size: 3
        class_prediction_bias_init: -4.599999904632568
      }
    }
    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: 9.99999993922529e-09
        iou_threshold: 0.6000000238418579
        max_detections_per_class: 100
        max_total_detections: 100
      }
      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: true
  }
```

Build the model

Recall that you imported model_builder.

- You'll use `model_builder` to build the model according to the configurations that you have just downloaded and customized.

Exercise 5.4: Build the custom model

model_builder

model_builder has a function `build`:

```
def build(model_config, is_training, add_summaries=True):
```

- model_config: Set this to the model configuration that you just customized.
- is_training: Set this to True.
- You can keep the default value for the remaining parameter.

```
### START CODE HERE (Replace instances of `None` with your code) ###
detection_model = model_builder.build(model_config=model_config, is_training=True)
### END CODE HERE ###

print(type(detection_model))
```

```
    <class 'object_detection.meta_architectures.ssd_meta_arch.SSDMetaArch'>
```

Expected Output:

```
<class 'object_detection.meta_architectures.ssd_meta_arch.SSDMetaArch'>
```

Restore weights from your checkpoint

Now, you will selectively restore weights from your checkpoint.

- Your end goal is to create a custom model which reuses parts of, but not all of the layers of RetinaNet (currently stored in the variable `detection_model`.)
  - The parts of RetinaNet that you want to reuse are:
    - Feature extraction layers
    - Bounding box regression prediction layer
  - The part of RetinaNet that you will not want to reuse is the classification prediction layer (since you will define and train your own classification layer specific to zombies).
  - For the parts of RetinaNet that you want to reuse, you will also restore the weights from the checkpoint that you selected.

Inspect the detection_model

First, take a look at the type of the detection_model and its Python class.

```
# Run this to check the type of detection_model
detection_model
```

```
    <object_detection.meta_architectures.ssd_meta_arch.SSDMetaArch at 0x7f3252f11610>
```

Find the source code for detection_model

You'll see that the type of the model is `object_detection.meta_architectures.ssd_meta_arch.SSDMetaArch`. Please practice some detective work and open up the source code for this class in GitHub repository. Recall that at the start of this assignment, you cloned from this repository: TensorFlow Models.

- Navigate through these subfolders: models -> research -> object_detection.
    - If you get stuck, go to this link: object_detection
- Take a look at this 'object_detection' folder and look for the remaining folders to navigate based on the class type of detection_model: object_detection.meta_architectures.ssd_meta_arch.SSDMetaArch
    - Hopefully you'll find the meta_architectures folder, and within it you'll notice a file named `ssd_meta_arch.py`.
    - Please open and view this ssd_meta_arch.py file.

View the variables in detection_model

Now, check the class variables that are in `detection_model`.

```
vars(detection_model)
```

```
{'_activity_regularizer': None,
 '_add_background_class': True,
 '_add_summaries': True,
 '_anchor_generator': <object_detection.anchor_generators.multiscale_grid_anchor_generator.
 '_anchors': None,
 '_auto_track_sub_layers': True,
 '_autocast': True,
 '_batched_prediction_tensor_names': ListWrapper([]),
 '_box_coder': <object_detection.box_coders.faster_rcnn_box_coder.FasterRcnnBoxCoder at 0x7
 '_box_predictor': <object_detection.predictors.convolutional_keras_box_predictor.WeightSha
 '_build_input_shape': None,
 '_callable_losses': [],
 '_classification_loss': <object_detection.core.losses.SigmoidFocalClassificationLoss at 0x
 '_classification_loss_weight': 1.0,
 '_compute_dtype_object': tf.float32,
 '_default_training_arg': None,
 '_dtype_policy': <Policy "float32">,
 '_dynamic': False,
 '_equalization_loss_config': EqualizationLossConfig(weight=0.0, exclude_prefixes=[]),
 '_expected_loss_weights_fn': None,
 '_expects_mask_arg': False,
 '_expects_training_arg': False,
 '_explicit_background_class': False,
 '_extract_features_scope': 'ResNet50V1_FPN',
 '_feature_extractor': <object_detection.models.ssd_resnet_v1_fpn_keras_feature_extractor.S
 '_freeze_batchnorm': True,
 '_groundtruth_lists': DictWrapper({}),
 '_hard_example_miner': None,
 '_image_resizer_fn': functools.partial(<function resize_image at 0x7f3255a8c0e0>, new_heig
 '_implicit_example_weight': 1.0,
 '_inbound_nodes_value': [],
 '_initial_weights': None,
 '_inplace_batchnorm_update': True,
 '_input_spec': None,
 '_instrumented_keras_api': True,
 '_instrumented_keras_layer_class': True,
 '_instrumented_keras_model_class': False,
 '_is_training': True,
 '_localization_loss': <object_detection.core.losses.WeightedSmoothL1LocalizationLoss at 0x
 '_localization_loss_weight': 1.0,
 '_losses': [],
 '_metrics': [],
 '_metrics_lock': <unlocked _thread.lock object at 0x7f3252f0db40>,
 '_name': 'ssd_meta_arch',
 '_non_max_suppression_fn': functools.partial(<function batch_multiclass_non_max_suppressio
 '_non_trainable_weights': [],
 '_normalize_loc_loss_by_codesize': True,
 '_normalize_loss_by_num_matches': True,
 '_num_classes': 1,
 '_obj_reference_counts_dict': ObjectIdentityDictionary({<_ObjectIdentityWrapper wrapping 1
 '_outbound_nodes_value': [],
 '_parallel_iterations': 16,
 '_preserve_input_structure_in_config': False,
 '_random_example_sampler': None,
 '_return_raw_detections_during_predict': False,
 '_saved_model_inputs_spec': None,
 '_score_conversion_fn': <function object_detection.builders.post_processing_builder._score
```

You'll see that detection_model contains several variables:

Two of these will be relevant to you:

```
...
_box_predictor': <object_detection.predictors.convolutional_keras_box_predictor.WeightSharedConvolutionalBox
...
_feature_extractor': <object_detection.models.ssd_resnet_v1_fpn_keras_feature_extractor.SSDResNet50V1FpnKera
```

Inspect `_feature_extractor`

Take a look at the ssd_meta_arch.py code.

```
# Line 302
feature_extractor: a SSDFeatureExtractor object.
```

Also

```
# Line 380
self._feature_extractor = feature_extractor
```

So `detection_model._feature_extractor` is a feature extractor, which you will want to reuse for your zombie detector model.

Inspect `_box_predictor`

- View the ssd_meta_arch.py file (which is the source code for detection_model)
- Notice that in the **init** constructor for class SSDMetaArch(model.DetectionModel),

```
...
box_predictor: a box_predictor.BoxPredictor object
```

```
    ...
    self._box_predictor = box_predictor
```

Inspect _box_predictor

Please take a look at the class type of `detection_model._box_predictor`

```
# view the type of _box_predictor
detection_model._box_predictor
```

```
<object_detection.predictors.convolutional_keras_box_predictor.WeightSharedConvolutionalBoxPre
```

You'll see that the class type of _box_predictor is

```
object_detection.predictors.convolutional_keras_box_predictor.WeightSharedConvolutionalBoxPredictor
```

You can navigate through the GitHub repository to this path:

- [objection_detection/predictors]()
- Notice that there is a file named convolutional_keras_box_predictor.py. Please open that file.

View variables in `_box_predictor`

Also view the variables contained in _box_predictor:

```
vars(detection_model._box_predictor)
```

```
{'_activity_regularizer': None,
 '_additional_projection_layers': ListWrapper([]),
 '_apply_batch_norm': True,
 '_apply_conv_hyperparams_pointwise': False,
 '_auto_track_sub_layers': True,
 '_autocast': True,
 '_base_tower_layers_for_heads': DictWrapper({'box_encodings': ListWrapper([]), 'class_pred
 '_box_prediction_head': <object_detection.predictors.heads.keras_box_head.WeightSharedConv
 '_build_input_shape': None,
 '_callable_losses': [],
 '_compute_dtype_object': tf.float32,
 '_conv_hyperparams': <object_detection.builders.hyperparams_builder.KerasLayerHyperparams
 '_default_training_arg': None,
 '_depth': 256,
 '_dtype_policy': <Policy "float32">,
 '_dynamic': False,
 '_expects_mask_arg': True,
 '_expects_training_arg': True,
 '_freeze_batchnorm': True,
 '_head_scope_conv_layers': DictWrapper({}),
 '_inbound_nodes_value': [],
 '_initial_weights': None,
 '_inplace_batchnorm_update': False,
 '_input_spec': None,
 '_instrumented_keras_api': True,
 '_instrumented_keras_layer_class': True,
 '_instrumented_keras_model_class': False,
 '_is_training': True,
 '_kernel_size': 3,
 '_losses': [],
 '_metrics': [],
 '_metrics_lock': <unlocked _thread.lock object at 0x7f325590d480>,
 '_name': 'WeightSharedConvolutionalBoxPredictor',
 '_non_trainable_weights': [],
 '_num_classes': 1,
 '_num_layers_before_predictor': 4,
 '_obj_reference_counts_dict': ObjectIdentityDictionary({<_ObjectIdentityWrapper wrapping T
 '_outbound_nodes_value': [],
 '_prediction_heads': DictWrapper({'class_predictions_with_background': <object_detection.p
 '_preserve_input_structure_in_config': False,
 '_saved_model_inputs_spec': None,
 '_self_name_based_restores': set(),
 '_self_saveable_object_factories': {},
 '_self_setattr_tracking': True,
 '_self_tracked_trackables': [<object_detection.predictors.heads.keras_box_head.WeightShare
  DictWrapper({'class_predictions_with_background': <object_detection.predictors.heads.kera
  ListWrapper(['class_predictions_with_background']),
  ListWrapper([]),
  DictWrapper({'box_encodings': ListWrapper([]), 'class_predictions_with_background': ListW
  DictWrapper({})],
 '_self_unconditional_checkpoint_dependencies': [TrackableReference(name='_box_prediction_h
  TrackableReference(name='_prediction_heads', ref=DictWrapper({'class_predictions_with_bac
  TrackableReference(name='_sorted_head_names', ref=ListWrapper(['class_predictions_with_ba
  TrackableReference(name='_additional_projection_layers', ref=ListWrapper([])),
  TrackableReference(name='_base_tower_layers_for_heads', ref=DictWrapper({'box_encodings':
  TrackableReference(name='_head_scope_conv_layers', ref=DictWrapper({}))],
 '_self_unconditional_deferred_dependencies': {},
 '_self_unconditional_dependency_names': {'_additional_projection_layers': ListWrapper([]),
```

Among the variables listed, a few will be relevant to you:

```
...
_base_tower_layers_for_heads
...
_box_prediction_head
...
_prediction_heads
```

In the source code for convolutional_keras_box_predictor.py that you just opened, look at the source code to get a sense for what these three variables represent.

Inspect `base_tower_layers_for_heads`

If you look at the convolutional_keras_box_predictor.py file, you'll notice this:

```
# line 302
self._base_tower_layers_for_heads = {
        BOX_ENCODINGS: [],
        CLASS_PREDICTIONS_WITH_BACKGROUND: [],
    }
```

- `base_tower_layers_for_heads` is a dictionary with two key-value pairs.
    - `BOX_ENCODINGS` : points to a list of layers
    - `CLASS_PREDICTIONS_WITH_BACKGROUND` : points to a list of layers

    - If you scan the code, you'll see that for both of these, the lists are filled with all layers that appear BEFORE the prediction layer.

      ```
      # Line 377
      # Stack the base_tower_layers in the order of conv_layer, batch_norm_layer
      # and activation_layer
      base_tower_layers = []
      for i in range(self._num_layers_before_predictor):
      ```

So `detection_model.box_predictor._base_tower_layers_for_heads` contains:

- The layers for the prediction before the final bounding box prediction
- The layers for the prediction before the final class prediction.

[illegible]

Inspect `_box_prediction_head`

If you again look at convolutional_keras_box_predictor.py file, you'll see this

```
# Line 248
box_prediction_head: The head that predicts the boxes.
```

So `detection_model.box_predictor._box_prediction_head` points to the bounding box prediction layer, which you'll want to use for your model.

Inspect `_prediction_heads`

If you again look at convolutional_keras_box_predictor.py file, you'll see this

```
# Line 121
self._prediction_heads = {
        BOX_ENCODINGS: box_prediction_heads,
        CLASS_PREDICTIONS_WITH_BACKGROUND: class_prediction_heads,
    }
```

You'll also see this docstring

```
# Line 83
class_prediction_heads: A list of heads that predict the classes.
```

So `detection_model.box_predictor._prediction_heads` is a dictionary that points to both prediction layers:

- The layer that predicts the bounding boxes
- The layer that predicts the class (category).

Which layers will you reuse?

Remember that you are reusing the model for its feature extraction and bounding box detection.

- You will create your own classification layer and train it on zombie images.
- So you won't need to reuse the class prediction layer of `detection_model`.

Define checkpoints for desired layers

You will now isolate the layers of `detection_model` that you wish to reuse so that you can restore the weights to just those layers.

- First, define checkpoints for the box predictor
- Next, define checkpoints for the model, which will point to this box predictor checkpoint as well as the feature extraction layers.

Please use tf.train.Checkpoint.

As a reminder of how to use tf.train.Checkpoint:

```
tf.train.Checkpoint(
    **kwargs
)
```

Pretend that `detection_model` contains these variables for which you want to restore weights:

- `detection_model._ice_cream_sundae`
- 'detection_model._pies._apple_pie`
- 'detection_model._pies._pecan_pie`

Notice that the pies are nested within `._pies`.

If you just want the ice cream sundae and apple pie variables (and not the pecan pie) then you can do the following:

```
tmp_pies_checkpoint = tf.train.Checkpoint(
  _apple_pie = detection_model._pies._apple_pie
)
```

Next, in order to connect these together in a node graph, do this:

```
tmp_model_checkpoint = tf.train.Checkpoint(
  _pies = tmp_pies_checkpoint,
  _ice_cream_sundae = detection_model._ice_cream_sundae
)
```

Finally, define a checkpoint that uses the key `model` and takes in the tmp_model_checkpoint.

```
checkpoint = tf.train.Checkpoint(
  model = tmp_model_checkpoint
)
```

You'll then be ready to restore the weights from the checkpoint that you downloaded.

Try this out step by step!

Exercise 6.1: Define Checkpoints for the box predictor

- Please define `box_predictor_checkpoint` to be checkpoint for these two layers of the `detection_model`'s box predictor:
    - The base tower layer (the layers the precede both the class prediction and bounding box prediction layers).
    - The box prediction head (the prediction layer for bounding boxes).
- Note, you won't include the class prediction layer.

```
### START CODE HERE ###

tmp_box_predictor_checkpoint = tf.compat.v2.train.Checkpoint( # tf.compat.v2.
    _base_tower_layers_for_heads=detection_model._box_predictor._base_tower_layers_for_heads,
    # _prediction_heads=detection_model._box_predictor._prediction_heads,
    #    (i.e., the classification head that we *will not* restore)
    _box_prediction_head=detection_model._box_predictor._box_prediction_head,
    )



### END CODE HERE
```

```
# Check the datatype of this checkpoint
type(tmp_box_predictor_checkpoint)

# Expected output:
# tensorflow.python.training.tracking.util.Checkpoint
```

```
     tensorflow.python.training.tracking.util.Checkpoint
```

```
# Check the variables of this checkpoint
vars(tmp_box_predictor_checkpoint)
```

```
     {'_attached_dependencies': None,
      '_base_tower_layers_for_heads': DictWrapper({'box_encodings': ListWrapper([]), 'class_predict
      '_box_prediction_head': <object_detection.predictors.heads.keras_box_head.WeightSharedConvolu
      '_save_assign_op': None,
      '_save_counter': None,
      '_saver': <tensorflow.python.training.tracking.util.TrackableSaver at 0x7f32405ffad0>,
      '_self_name_based_restores': set(),
      '_self_saveable_object_factories': {},
      '_self_setattr_tracking': True,
      '_self_unconditional_checkpoint_dependencies': [TrackableReference(name='_base_tower_layers_f
       TrackableReference(name='_box_prediction_head', ref=<object_detection.predictors.heads.keras
      '_self_unconditional_deferred_dependencies': {},
      '_self_unconditional_dependency_names': {'_base_tower_layers_for_heads': DictWrapper({'box_en
       '_box_prediction_head': <object_detection.predictors.heads.keras_box_head.WeightSharedConvol
      '_self_update_uid': -1}
```

Expected output

You should expect to see a list of variables that include the following:

```
'_base_tower_layers_for_heads': DictWrapper({'box_encodings': ListWrapper([]), 'class_predictions_with_backg
'_box_prediction_head': <object_detection.predictors.heads.keras_box_head.WeightSharedConvolutionalBoxHead a
 ...
```

Exercise 6.2: Define the temporary model checkpoint**

Now define `tmp_model_checkpoint` so that it points to these two layers:

- The feature extractor of the detection model.
- The temporary box predictor checkpoint that you just defined.

```
### START CODE HERE ###
tmp_model_checkpoint = tf.compat.v2.train.Checkpoint(
          _feature_extractor=detection_model._feature_extractor,
          _box_predictor=tmp_box_predictor_checkpoint)

tmp_model_checkpoint = tf.compat.v2.train.Checkpoint(model=tmp_model_checkpoint)



### END CODE HERE ###


# Check the datatype of this checkpoint
type(tmp_model_checkpoint)

# Expected output
# tensorflow.python.training.tracking.util.Checkpoint
```

```
    tensorflow.python.training.tracking.util.Checkpoint
```

```
# Check the vars of this checkpoint
vars(tmp_model_checkpoint)
```

```
    {'_attached_dependencies': None,
     '_save_assign_op': None,
     '_save_counter': None,
     '_saver': <tensorflow.python.training.tracking.util.TrackableSaver at 0x7f32405ff490>,
     '_self_name_based_restores': set(),
     '_self_saveable_object_factories': {},
     '_self_setattr_tracking': True,
     '_self_unconditional_checkpoint_dependencies': [TrackableReference(name='model', ref=<tensorf
     '_self_unconditional_deferred_dependencies': {},
     '_self_unconditional_dependency_names': {'model': <tensorflow.python.training.tracking.util.C
     '_self_update_uid': -1,
     'model': <tensorflow.python.training.tracking.util.Checkpoint at 0x7f3240591a90>}
```

Expected output

Among the variables of this checkpoint, you should see:

```
 '_box_predictor': <tensorflow.python.training.tracking.util.Checkpoint at 0x7fefac044a20>,
  '_feature_extractor': <object_detection.models.ssd_resnet_v1_fpn_keras_feature_extractor.SSDResNet50V1FpnKe
```

Exercise 6.3: Restore the checkpoint

You can now restore the checkpoint.

First, find and set the `checkpoint_path`

- checkpoint_path:
    - Using the "files" browser in the left side of Colab, navigate to `models -> research -> object_detection -> test_data`.
    - If you completed the previous code cell that downloads and moves the checkpoint, you'll see a subfolder named "checkpoint".
        - The 'checkpoint' folder contains three files:
            - checkpoint
            - ckpt-0.data-00000-of-00001
            - ckpt-0.index
        - Please set checkpoint_path to the path to the full path `models/.../ckpt-0`
            - Notice that you don't want to include a file extension after `ckpt-0`.
        - **IMPORTANT**: Please don't set the path to include the `.index` extension in the checkpoint file name.
            - If you do set it to `ckpt-0.index`, there won't be any immediate error message, but later during training, you'll notice that your model's loss doesn't improve, which means that the pre-trained weights were not restored properly.

Next, define one last checkpoint using `tf.train.Checkpoint()`.

- For the single keyword argument,
    - Set the key as `model=`
    - Set the value to your temporary model checkpoint that you just defined.
- **IMPORTANT**: You'll need to set the keyword argument as `model=` and not something else like `detection_model=`.
- If you set this keyword argument to anything else, it won't show an immmediate error, but when you train your model on the zombie images, your model loss will not decrease (your model will not learn).

Finally, call this checkpoint's `.restore()` function, passing in the path to the checkpoint.

```
### START CODE HERE ###

checkpoint_path = 'models/research/object_detection/test_data/checkpoint/ckpt-0'


# Define a checkpoint that sets `model= None
checkpoint = tf.compat.v2.train.Checkpoint(model=tmp_model_checkpoint)
```

```
# Restore the checkpoint to the checkpoint path
checkpoint.restore(checkpoint_path).expect_partial()

### END CODE HERE ###
```

```
<tensorflow.python.training.tracking.util.CheckpointLoadStatus at 0x7f3240529250>
```

```
tf.keras.backend.clear_session()

print('Building model and restoring weights for fine-tuning...', flush=True)
num_classes = 1
pipeline_config = 'models/research/object_detection/configs/tf2/ssd_resnet50_v1_fpn_640x640_coco17_t
checkpoint_path = 'models/research/object_detection/test_data/checkpoint/ckpt-0'

# Load pipeline config and build a detection model.
#
# Since we are working off of a COCO architecture which predicts 90
# class slots by default, we override the `num_classes` field here to be just
# one (for our new rubber ducky class).
configs = config_util.get_configs_from_pipeline_file(pipeline_config)
model_config = configs['model']
model_config.ssd.num_classes = num_classes
model_config.ssd.freeze_batchnorm = True
detection_model = model_builder.build(
      model_config=model_config, is_training=True)

# Set up object-based checkpoint restore --- RetinaNet has two prediction
# `heads` --- one for classification, the other for box regression.  We will
# restore the box regression head but initialize the classification head
# from scratch (we show the omission below by commenting out the line that
# we would add if we wanted to restore both heads)
fake_box_predictor = tf.compat.v2.train.Checkpoint(
    _base_tower_layers_for_heads=detection_model._box_predictor._base_tower_layers_for_heads,
    # _prediction_heads=detection_model._box_predictor._prediction_heads,
    #    (i.e., the classification head that we *will not* restore)
    _box_prediction_head=detection_model._box_predictor._box_prediction_head,
    )
fake_model = tf.compat.v2.train.Checkpoint(
          _feature_extractor=detection_model._feature_extractor,
          _box_predictor=fake_box_predictor)
ckpt = tf.compat.v2.train.Checkpoint(model=fake_model)
ckpt.restore(checkpoint_path).expect_partial()

# Run model through a dummy image so that variables are created
image, shapes = detection_model.preprocess(tf.zeros([1, 640, 640, 3]))
prediction_dict = detection_model.predict(image, shapes)
_ = detection_model.postprocess(prediction_dict, shapes)
print('Weights restored!')
```

```
Building model and restoring weights for fine-tuning...
Weights restored!
```

Exercise 7: Run a dummy image to generate the model variables

Run a dummy image through the model so that variables are created. We need to select the trainable variables later in Exercise 9 and right now, it is still empty. Try running `len(detection_model.trainable_variables)` in a code cell and you will get `0`. We will pass in a dummy image through the forward pass to create these variables.

Recall that `detection_model` is an object of type [object_detection.meta_architectures.ssd_meta_arch.SSDMetaArch]()

Important methods that are available in the `detection_model` object are:

- [preprocess()]():
  - takes in a tensor representing an image and returns
  - returns `image, shapes`
  - For the dummy image, you can declare a [tensor of zeros]() that has a shape that the `preprocess()` method can accept (i.e. [batch, height, width, channels]).
  - Remember that your images have dimensions 640 x 640 x 3.
  - You can pass in a batch of 1 when making the dummy image.
- [predict()]()
  - takes in `image, shapes` which are created by the `preprocess()` function call.
  - returns a prediction in a Python dictionary
  - this will pass the dummy image through the forward pass of the network and create the model variables
- [postprocess()]()
  - Takes in the prediction_dict and shapes
  - returns a dictionary of post-processed predictions of detected objects ("detections").

Note: Please use the recommended variable names, which include the prefix `tmp_`, since these variables won't be used later, but you'll define similarly-named variables later for predicting on actual zombie images.

```
### START CODE HERE (Replace instances of `None` with your code)###

# use the detection model's `preprocess()` method and pass a dummy image
tmp_image, tmp_shapes = detection_model.preprocess(tf.zeros([1, 640, 640, 3]))

# run a prediction with the preprocessed image and shapes
tmp_prediction_dict = detection_model.predict(tmp_image, tmp_shapes)

# postprocess the predictions into final detections
tmp_detections = detection_model.postprocess(tmp_prediction_dict, tmp_shapes)

### END CODE HERE ###

print('Weights restored!')
```

```
    Weights restored!
```

```
# Test Code:
assert len(detection_model.trainable_variables) > 0, "Please pass in a dummy image to create the tra
```

```
print(detection_model.weights[0].shape)
print(detection_model.weights[231].shape)
print(detection_model.weights[462].shape)
```

```
     (3, 3, 256, 24)
     (512,)
     (256,)
```

Expected Output:

```
(3, 3, 256, 24)
(512,)
(256,)
```

Eager mode custom training loop

With the data and model now setup, you can now proceed to configure the training.

Exercise 8: Set training hyperparameters

Set an appropriate learning rate and optimizer for the training.

- batch_size: you can use 4
  - You can increase the batch size up to 5, since you have just 5 images for training.
- num_batches: You can use 100
  - You can increase the number of batches but the training will take longer to complete.
- learning_rate: You can use 0.01
  - When you run the training loop later, notice how the initial loss INCREASES` before decreasing.
  - You can try a lower learning rate to see if you can avoid this increased loss.
- optimizer: you can use tf.keras.optimizers.SGD
  - Set the learning rate
  - Set the momentum to 0.9

Training will be fairly quick, so we do encourage you to experiment a bit with these hyperparameters!

```
tf.keras.backend.set_learning_phase(True)

### START CODE HERE (Replace instances of `None` with your code)###

# set the batch_size
batch_size = 4

# set the number of batches
num_batches = 100

# Set the learning rate
learning_rate = 0.01
learning_rate = 0.01

# set the optimizer and pass in the learning_rate
optimizer = tf.keras.optimizers.SGD(learning_rate=learning_rate, momentum=0.9)

### END CODE HERE ###
```

```
/usr/local/lib/python3.7/dist-packages/tensorflow/python/keras/backend.py:435: UserWarning: `t
  warnings.warn('`tf.keras.backend.set_learning_phase` is deprecated and '
```

Choose the layers to fine-tune

To make use of transfer learning and pre-trained weights, you will train just certain parts of the detection model, namely, the last prediction layers.

- Please take a minute to inspect the layers of `detection_model`.

```
# Inspect the layers of detection_model
for i,v in enumerate(detection_model.trainable_variables):
    print(f"i: {i} \t name: {v.name} \t shape:{v.shape} \t dtype={v.dtype}")
```

```
i: 211   name: conv4_block5_1_bn/beta:0          shape:(256,)    dtype=<dtype: 'float32'>
i: 212   name: conv4_block5_2_conv/kernel:0      shape:(3, 3, 256, 256)          dtype=<dty
i: 213   name: conv4_block5_2_bn/gamma:0         shape:(256,)    dtype=<dtype: 'float32'>
i: 214   name: conv4_block5_2_bn/beta:0          shape:(256,)    dtype=<dtype: 'float32'>
i: 215   name: conv4_block5_3_conv/kernel:0      shape:(1, 1, 256, 1024)         dtype=<dty
i: 216   name: conv4_block5_3_bn/gamma:0         shape:(1024,)   dtype=<dtype: 'float32'>
i: 217   name: conv4_block5_3_bn/beta:0          shape:(1024,)   dtype=<dtype: 'float32'>
i: 218   name: conv4_block6_1_conv/kernel:0      shape:(1, 1, 1024, 256)         dtype=<dty
i: 219   name: conv4_block6_1_bn/gamma:0         shape:(256,)    dtype=<dtype: 'float32'>
i: 220   name: conv4_block6_1_bn/beta:0          shape:(256,)    dtype=<dtype: 'float32'>
i: 221   name: conv4_block6_2_conv/kernel:0      shape:(3, 3, 256, 256)          dtype=<dty
i: 222   name: conv4_block6_2_bn/gamma:0         shape:(256,)    dtype=<dtype: 'float32'>
i: 223   name: conv4_block6_2_bn/beta:0          shape:(256,)    dtype=<dtype: 'float32'>
i: 224   name: conv4_block6_3_conv/kernel:0      shape:(1, 1, 256, 1024)         dtype=<dty
i: 225   name: conv4_block6_3_bn/gamma:0         shape:(1024,)   dtype=<dtype: 'float32'>
i: 226   name: conv4_block6_3_bn/beta:0          shape:(1024,)   dtype=<dtype: 'float32'>
i: 227   name: conv5_block1_1_conv/kernel:0      shape:(1, 1, 1024, 512)         dtype=<dty
i: 228   name: conv5_block1_1_bn/gamma:0         shape:(512,)    dtype=<dtype: 'float32'>
i: 229   name: conv5_block1_1_bn/beta:0          shape:(512,)    dtype=<dtype: 'float32'>
i: 230   name: conv5_block1_2_conv/kernel:0      shape:(3, 3, 512, 512)          dtype=<dty
i: 231   name: conv5_block1_2_bn/gamma:0         shape:(512,)    dtype=<dtype: 'float32'>
i: 232   name: conv5_block1_2_bn/beta:0          shape:(512,)    dtype=<dtype: 'float32'>
i: 233   name: conv5_block1_0_conv/kernel:0      shape:(1, 1, 1024, 2048)        dtype=<dty
i: 234   name: conv5_block1_3_conv/kernel:0      shape:(1, 1, 512, 2048)         dtype=<dty
i: 235   name: conv5_block1_0_bn/gamma:0         shape:(2048,)   dtype=<dtype: 'float32'>
i: 236   name: conv5_block1_0_bn/beta:0          shape:(2048,)   dtype=<dtype: 'float32'>
i: 237   name: conv5_block1_3_bn/gamma:0         shape:(2048,)   dtype=<dtype: 'float32'>
i: 238   name: conv5_block1_3_bn/beta:0          shape:(2048,)   dtype=<dtype: 'float32'>
i: 239   name: conv5_block2_1_conv/kernel:0      shape:(1, 1, 2048, 512)         dtype=<dty
i: 240   name: conv5_block2_1_bn/gamma:0         shape:(512,)    dtype=<dtype: 'float32'>
i: 241   name: conv5_block2_1_bn/beta:0          shape:(512,)    dtype=<dtype: 'float32'>
i: 242   name: conv5_block2_2_conv/kernel:0      shape:(3, 3, 512, 512)          dtype=<dty
i: 243   name: conv5_block2_2_bn/gamma:0         shape:(512,)    dtype=<dtype: 'float32'>
i: 244   name: conv5_block2_2_bn/beta:0          shape:(512,)    dtype=<dtype: 'float32'>
i: 245   name: conv5_block2_3_conv/kernel:0      shape:(1, 1, 512, 2048)         dtype=<dty
i: 246   name: conv5_block2_3_bn/gamma:0         shape:(2048,)   dtype=<dtype: 'float32'>
i: 247   name: conv5_block2_3_bn/beta:0          shape:(2048,)   dtype=<dtype: 'float32'>
i: 248   name: conv5_block3_1_conv/kernel:0      shape:(1, 1, 2048, 512)         dtype=<dty
i: 249   name: conv5_block3_1_bn/gamma:0         shape:(512,)    dtype=<dtype: 'float32'>
```

```
i: 250   name: conv5_block3_1_bn/beta:0         shape:(512,)    dtype=<dtype: 'float32'>
i: 251   name: conv5_block3_2_conv/kernel:0      shape:(3, 3, 512, 512)         dtype=<dty
i: 252   name: conv5_block3_2_bn/gamma:0         shape:(512,)    dtype=<dtype: 'float32'>
i: 253   name: conv5_block3_2_bn/beta:0          shape:(512,)    dtype=<dtype: 'float32'>
i: 254   name: conv5_block3_3_conv/kernel:0      shape:(1, 1, 512, 2048)        dtype=<dty
i: 255   name: conv5_block3_3_bn/gamma:0         shape:(2048,)   dtype=<dtype: 'float32'>
i: 256   name: conv5_block3_3_bn/beta:0          shape:(2048,)   dtype=<dtype: 'float32'>
i: 257   name: ResNet50V1_FPN/FeatureMaps/top_down/projection_3/kernel:0        shape:(1,
i: 258   name: ResNet50V1_FPN/FeatureMaps/top_down/projection_3/bias:0   shape:(256,)    dt
i: 259   name: ResNet50V1_FPN/FeatureMaps/top_down/projection_2/kernel:0        shape:(1,
i: 260   name: ResNet50V1_FPN/FeatureMaps/top_down/projection_2/bias:0   shape:(256,)    dt
i: 261   name: ResNet50V1_FPN/FeatureMaps/top_down/projection_1/kernel:0        shape:(1,
i: 262   name: ResNet50V1_FPN/FeatureMaps/top_down/projection_1/bias:0   shape:(256,)    dt
i: 263   name: ResNet50V1_FPN/FeatureMaps/top_down/smoothing_2_conv/kernel:0    shape:(3,
i: 264   name: ResNet50V1_FPN/FeatureMaps/top_down/smoothing_2_batchnorm/gamma:0        sh
i: 265   name: ResNet50V1_FPN/FeatureMaps/top_down/smoothing_2_batchnorm/beta:0         sh
i: 266   name: ResNet50V1_FPN/FeatureMaps/top_down/smoothing_1_conv/kernel:0    shape:(3,
i: 267   name: ResNet50V1_FPN/FeatureMaps/top_down/smoothing_1_batchnorm/gamma:0        sh
i: 268   name: ResNet50V1_FPN/FeatureMaps/top_down/smoothing_1_batchnorm/beta:0         sh
```

Notice that there are some layers whose names are prefixed with the following:

```
WeightSharedConvolutionalBoxPredictor/WeightSharedConvolutionalBoxHead
...
WeightSharedConvolutionalBoxPredictor/WeightSharedConvolutionalClassHead
...
WeightSharedConvolutionalBoxPredictor/BoxPredictionTower
...
WeightSharedConvolutionalBoxPredictor/ClassPredictionTower
...
```

Among these, which do you think are the prediction layers at the "end" of the model?

- Recall that when inspecting the source code to restore the checkpoints (convolutional_keras_box_predictor.py) you noticed that:
    - `_base_tower_layers_for_heads` : refers to the layers that are placed right before the prediction layer
    - `_box_prediction_head` refers to the prediction layer for the bounding boxes
    - `_prediction_heads` : refers to the set of prediction layers (both for classification and for bounding boxes)

So you can see that in the source code for this model, "tower" refers to layers that are before the prediction layer, and "head" refers to the prediction layers.

Exercise 9: Select the prediction layer variables

Based on inspecting the `detection_model.trainable_variables` , please select the prediction layer variables that you will fine tune:

- The bounding box head variables (which predict bounding box coordinates)
- The class head variables (which predict the class/category)

You have a few options for doing this:

- You can access them by their list index:

```
detection_model.trainable_variables[92]
```

- Alternatively, you can use string matching to select the variables:

```
tmp_list = []
for v in detection_model.trainable_variables:
if v.name.startswith('ResNet50V1_FPN/bottom_up_block5'):
  tmp_list.append(v)
```

Hint: There are a total of four variables that you want to fine tune.

```
### START CODE HERE (Replace instances of `None` with your code) ###

# define a list that contains the layers that you wish to fine tune
to_fine_tune = []
prefixes_to_train = [
  'WeightSharedConvolutionalBoxPredictor/WeightSharedConvolutionalBoxHead',
  'WeightSharedConvolutionalBoxPredictor/WeightSharedConvolutionalClassHead']
for var in detection_model.trainable_variables:
  if any([var.name.startswith(prefix) for prefix in prefixes_to_train]):
    to_fine_tune.append(var)

### END CODE HERE


# Test Code:

print(to_fine_tune[0].name)
print(to_fine_tune[2].name)
```

```
WeightSharedConvolutionalBoxPredictor/WeightSharedConvolutionalBoxHead/BoxPredictor/kernel:0
WeightSharedConvolutionalBoxPredictor/WeightSharedConvolutionalClassHead/ClassPredictor/kernel
```

Expected Output:

```
WeightSharedConvolutionalBoxPredictor/WeightSharedConvolutionalBoxHead/BoxPredictor/kernel:0
WeightSharedConvolutionalBoxPredictor/WeightSharedConvolutionalClassHead/ClassPredictor/kernel:0
```

Train your model

You'll define a function that handles training for one batch, which you'll later use in your training loop.

First, walk through these code cells to learn how you'll perform training using this model.

```
# Get a batch of your training images
g_images_list = train_image_tensors[0:2]
```

The `detection_model` is of class SSDMetaArch, and its source code shows that is has this function preprocess.

- This preprocesses the images so that they can be passed into the model (for training or prediction):

```
def preprocess(self, inputs):
  """Feature-extractor specific preprocessing.
  ...
  Args:
    inputs: a [batch, height_in, width_in, channels] float tensor representing
      a batch of images with values between 0 and 255.0.
  Returns:
    preprocessed_inputs: a [batch, height_out, width_out, channels] float
      tensor representing a batch of images.

    true_image_shapes: int32 tensor of shape [batch, 3] where each row is
      of the form [height, width, channels] indicating the shapes
      of true images in the resized images, as resized images can be padded
      with zeros.
```

```
# Use .preprocess to preprocess an image
g_preprocessed_image = detection_model.preprocess(g_images_list[0])
print(f"g_preprocessed_image type: {type(g_preprocessed_image)}")
print(f"g_preprocessed_image length: {len(g_preprocessed_image)}")
print(f"index 0 has the preprocessed image of shape {g_preprocessed_image[0].shape}")
print(f"index 1 has information about the image's true shape excluding padding: {g_preprocessed_imag
```

```
g_preprocessed_image type: <class 'tuple'>
g_preprocessed_image length: 2
index 0 has the preprocessed image of shape (1, 640, 640, 3)
index 1 has information about the image's true shape excluding padding: [[640 640   3]]
```

You can pre-process each image and save their outputs into two separate lists

- One list of the preprocessed images
- One list of the true shape for each preprocessed image

```
preprocessed_image_list = []
true_shape_list = []

for img in g_images_list:
    processed_img, true_shape = detection_model.preprocess(img)
    preprocessed_image_list.append(processed_img)
    true_shape_list.append(true_shape)

print(f"preprocessed_image_list is of type {type(preprocessed_image_list)}")
print(f"preprocessed_image_list has length {len(preprocessed_image_list)}")
print()
print(f"true_shape_list is of type {type(true_shape_list)}")
print(f"true_shape_list has length {len(true_shape_list)}")
print(f"true_shape_list has length {len(true_shape_list)}")
```

```
    preprocessed_image_list is of type <class 'list'>
    preprocessed_image_list has length 2

    true_shape_list is of type <class 'list'>
    true_shape_list has length 2
```

Make a prediction

The `detection_model` also has a `.predict` function. According to the source code for predict

```
  def predict(self, preprocessed_inputs, true_image_shapes):
    """Predicts unpostprocessed tensors from input tensor.
    This function takes an input batch of images and runs it through the forward
    pass of the network to yield unpostprocessesed predictions.
...
    Args:
      preprocessed_inputs: a [batch, height, width, channels] image tensor.

      true_image_shapes: int32 tensor of shape [batch, 3] where each row is
        of the form [height, width, channels] indicating the shapes
        of true images in the resized images, as resized images can be padded
        with zeros.

    Returns:
      prediction_dict: a dictionary holding "raw" prediction tensors:
        1) preprocessed_inputs: the [batch, height, width, channels] image
          tensor.
        2) box_encodings: 4-D float tensor of shape [batch_size, num_anchors,
          box_code_dimension] containing predicted boxes.
        3) class_predictions_with_background: 3-D float tensor of shape
          [batch_size, num_anchors, num_classes+1] containing class predictions
          (logits) for each of the anchors.  Note that this tensor *includes*
          background class predictions (at class index 0).
        4) feature_maps: a list of tensors where the ith tensor has shape
          [batch, height_i, width_i, depth_i].
        5) anchors: 2-D float tensor of shape [num_anchors, 4] containing
          the generated anchors in normalized coordinates.
        6) final_anchors: 3-D float tensor of shape [batch_size, num_anchors, 4]
          containing the generated anchors in normalized coordinates.
        If self._return_raw_detections_during_predict is True, the dictionary
        will also contain:
        7) raw_detection_boxes: a 4-D float32 tensor with shape
          [batch_size, self.max_num_proposals, 4] in normalized coordinates.
        8) raw_detection_feature_map_indices: a 3-D int32 tensor with shape
          [batch_size, self.max_num_proposals].
    """
```

Notice that `.predict` takes its inputs as tensors. If you tried to pass in the preprocessed images and true shapes, you'll get an error.

```
# Try to call `predict` and pass in lists; look at the error message
try:
    detection_model.predict(preprocessed_image_list, true_shape_list)
except AttributeError as e:
    print("Error message:", e)
```

```
Error message: 'list' object has no attribute 'get_shape'
```

But don't worry! You can check how to properly use `predict`:

- Notice that the source code documentation says that `preprocessed_inputs` and `true_image_shapes` are expected to be tensors and not lists of tensors.
- One way to turn a list of tensors into a tensor is to use [tf.concat](https://www.tensorflow.org/api_docs/python/tf/concat)

```
tf.concat(
    values, axis, name='concat'
)
```

```
# Turn a list of tensors into a tensor
preprocessed_image_tensor = tf.concat(preprocessed_image_list, axis=0)
true_shape_tensor = tf.concat(true_shape_list, axis=0)

print(f"preprocessed_image_tensor shape: {preprocessed_image_tensor.shape}")
print(f"true_shape_tensor shape: {true_shape_tensor.shape}")
```

```
preprocessed_image_tensor shape: (2, 640, 640, 3)
true_shape_tensor shape: (2, 3)
```

Now you can make predictions for the images. According to the source code, `predict` returns a dictionary containing the prediction information, including:

- The bounding box predictions
- The class predictions

```
# Make predictions on the images
prediction_dict = detection_model.predict(preprocessed_image_tensor, true_shape_tensor)

print("keys in prediction_dict:")
for key in prediction_dict.keys():
    print(key)
```

```
keys in prediction_dict:
preprocessed_inputs
feature_maps
anchors
final_anchors
box_encodings
class_predictions_with_background
```

Calculate loss

Now that your model has made its prediction, you want to compare it to the ground truth in order to calculate a loss.

- The `detection_model` has a loss function.

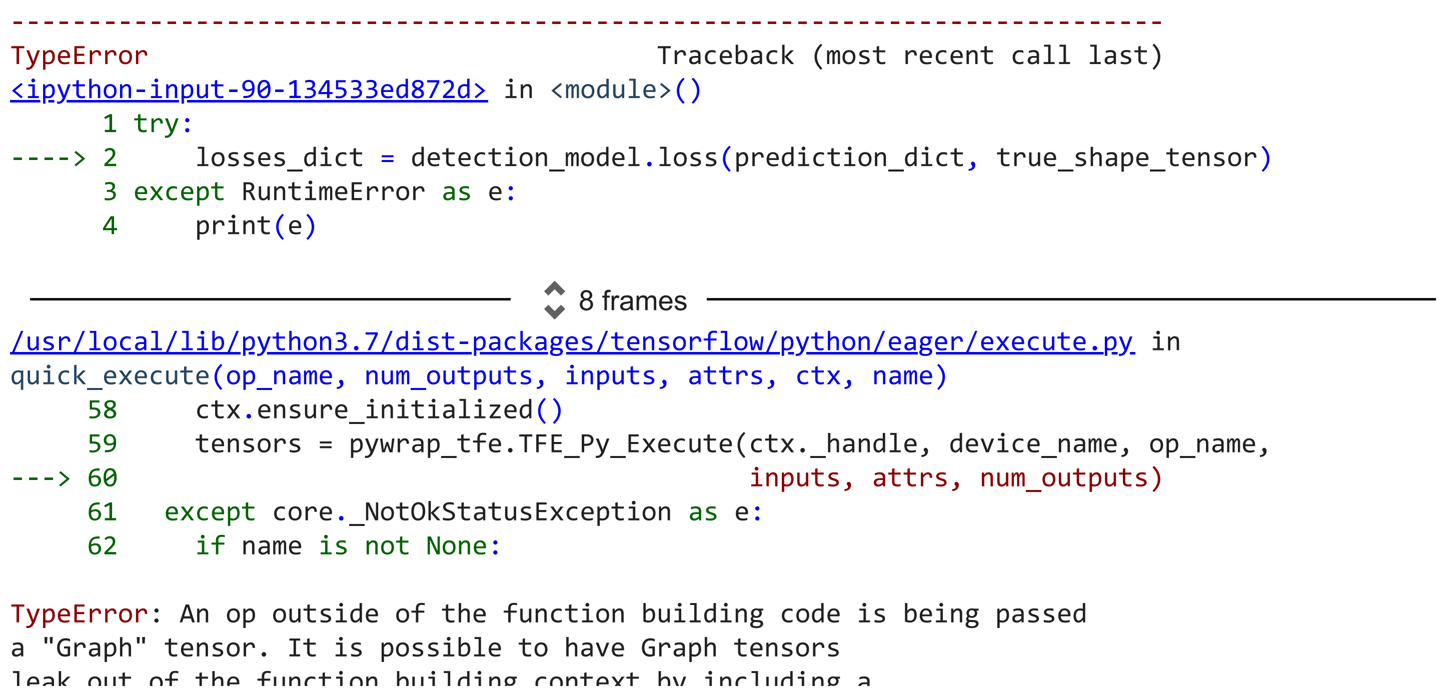
```
def loss(self, prediction_dict, true_image_shapes, scope=None):
  """Compute scalar loss tensors with respect to provided groundtruth.
  Calling this function requires that groundtruth tensors have been
  provided via the provide_groundtruth function.
  Args:
    prediction_dict: a dictionary holding prediction tensors with
      1) box_encodings: 3-D float tensor of shape [batch_size, num_anchors,
        box_code_dimension] containing predicted boxes.
      2) class_predictions_with_background: 3-D float tensor of shape
        [batch_size, num_anchors, num_classes+1] containing class predictions
        (logits) for each of the anchors. Note that this tensor *includes*
        background class predictions.
    true_image_shapes: int32 tensor of shape [batch, 3] where each row is
      of the form [height, width, channels] indicating the shapes
      of true images in the resized images, as resized images can be padded
      with zeros.
    scope: Optional scope name.
  Returns:
    a dictionary mapping loss keys (`localization_loss` and
      `classification_loss`) to scalar tensors representing corresponding loss
      values.
  """
```

It takes in:

- The prediction dictionary that comes from your call to `.predict()`.
- the true images shape that comes from your call to `.preprocess()` followed by the conversion from a list to a tensor.

Try calling `.loss`. You'll see an error message that you'll addres in order to run the `.loss` function.

```
try:
    losses_dict = detection_model.loss(prediction_dict, true_shape_tensor)
except RuntimeError as e:
    print(e)
```



```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-90-134533ed872d> in <module>()
      1 try:
----> 2     losses_dict = detection_model.loss(prediction_dict, true_shape_tensor)
      3 except RuntimeError as e:
      4     print(e)

8 frames
/usr/local/lib/python3.7/dist-packages/tensorflow/python/eager/execute.py in
quick_execute(op_name, num_outputs, inputs, attrs, ctx, name)
     58     ctx.ensure_initialized()
     59     tensors = pywrap_tfe.TFE_Py_Execute(ctx._handle, device_name, op_name,
---> 60                                         inputs, attrs, num_outputs)
     61   except core._NotOkStatusException as e:
     62     if name is not None:

TypeError: An op outside of the function building code is being passed
a "Graph" tensor. It is possible to have Graph tensors
leak out of the function building context by including a
```

This is giving an error about groundtruth_classes_list:

```
The graph tensor has name: groundtruth_classes_list:0
```

Notice in the docstring for `loss` (shown above), it says:

```
Calling this function requires that groundtruth tensors have been
    provided via the provide_groundtruth function.
```

So you'll first want to set the ground truth (true labels and true bounding boxes) before you calculate the loss.

- This makes sense, since the loss is comparing the prediction to the ground truth, and so the loss function needs to know the ground truth.

- Provide the ground truth

  The source code for providing the ground truth is located in the parent class of `SSDMetaArch`, `model.DetectionModel`.

- Here is the link to the code for provide_ground_truth

```
def provide_groundtruth(
      self,
      groundtruth_boxes_list,
      groundtruth_classes_list,
... # more parameters not show here
"""
    Args:
      groundtruth_boxes_list: a list of 2-D tf.float32 tensors of shape
        [num_boxes, 4] containing coordinates of the groundtruth boxes.
          Groundtruth boxes are provided in [y_min, x_min, y_max, x_max]
          format and assumed to be normalized and clipped
          relative to the image window with y_min <= y_max and x_min <= x_max.
      groundtruth_classes_list: a list of 2-D tf.float32 one-hot (or k-hot)
```

```
        tensors of shape [num_boxes, num_classes] containing the class targets
        with the 0th index assumed to map to the first non-background class.
"""
```

You'll set two parameters in `provide_ground_truth`:

- The true bounding boxes
- The true classes

```
# Get the ground truth bounding boxes
gt_boxes_list = gt_box_tensors[0:2]

# Get the ground truth class labels
gt_classes_list = gt_classes_one_hot_tensors[0:2]

# Provide the ground truth to the model
detection_model.provide_groundtruth(
            groundtruth_boxes_list=gt_boxes_list,
            groundtruth_classes_list=gt_classes_list)
```

Now you can calculate the loss

```
# Calculate the loss after you've provided the ground truth
losses_dict = detection_model.loss(prediction_dict, true_shape_tensor)

# View the loss dictionary
losses_dict = detection_model.loss(prediction_dict, true_shape_tensor)
print(f"loss dictionary keys: {losses_dict.keys()}")
print(f"localization loss {losses_dict['Loss/localization_loss']:.8f}")
print(f"classification loss {losses_dict['Loss/classification_loss']:.8f}")
```

```
    loss dictionary keys: dict_keys(['Loss/localization_loss', 'Loss/classification_loss'])
    localization loss 0.71293402
    classification loss 1.13724697
```

```
# Let's just reset the model so that you can practice setting it up yourself!
detection_model.provide_groundtruth(groundtruth_boxes_list=[], groundtruth_classes_list=[])
```

Exercise 10: Define the training step

Please complete the function below to set up one training step.

- Preprocess the images
- Make a prediction
- Calculate the loss (and make sure the loss function has the ground truth to compare with the prediction)
- Calculate the total loss:
  - `total_loss = localization_loss + classification_loss`
  - Note: this is different than the example code that you saw above
- Calculate gradients with respect to the variables you selected to train.
- Optimize the model's variables

```
# decorate with @tf.function for faster training (remember, graph mode!)
@tf.function
def train_step_fn(image_list,
                groundtruth_boxes_list,
                groundtruth_classes_list,
                model,
                optimizer,
                vars_to_fine_tune):
    """A single training iteration.

    Args:
      image_list: A list of [1, height, width, 3] Tensor of type tf.float32.
        Note that the height and width can vary across images, as they are
        reshaped within this function to be 640x640.
      groundtruth_boxes_list: A list of Tensors of shape [N_i, 4] with type
        tf.float32 representing groundtruth boxes for each image in the batch.
      groundtruth_classes_list: A list of Tensors of shape [N_i, num_classes]
        with type tf.float32 representing groundtruth boxes for each image in
        the batch.

    Returns:
      A scalar tensor representing the total loss for the input batch.
    """

    model.provide_groundtruth(
        groundtruth_boxes_list=groundtruth_boxes_list,
        groundtruth_classes_list=groundtruth_classes_list)

    with tf.GradientTape() as tape:
    ### START CODE HERE (Replace instances of `None` with your code) ###

        # Preprocess the images

        preprocessed_image_list = []
        true_shape_list = []

        for img in image_list:
            processed_img, true_shape = detection_model.preprocess(img)
            #print(true_shape)
            preprocessed_image_list.append(processed_img)
            true_shape_list.append(true_shape)


        preprocessed_image_tensor = tf.concat(preprocessed_image_list, axis=0)
        true_shape_tensor = tf.concat(true_shape_list, axis=0)

        # Make a prediction
        prediction_dict = model.predict(preprocessed_image_tensor, true_shape_tensor)

        # Calculate the total loss (sum of both losses)

        losses_dict = model.loss(prediction_dict, true_shape_tensor)
        total_loss = losses_dict['Loss/localization_loss'] + losses_dict['Loss/classification_loss'

        # Calculate the gradients
        gradients = tape.gradient(total_loss, vars_to_fine_tune)
        optimizer.apply_gradients(zip(gradients, vars_to_fine_tune))

        ### END CODE HERE ###

    return total_loss
```

Run the training loop

Run the training loop using the training step function that you just defined.

```
print('Start fine-tuning!', flush=True)

for idx in range(num_batches):
    # Grab keys for a random subset of examples
    all_keys = list(range(len(train_images_np)))
    random.shuffle(all_keys)
    example_keys = all_keys[:batch_size]

    # Get the ground truth
    gt_boxes_list = [gt_box_tensors[key] for key in example_keys]
    gt_classes_list = [gt_classes_one_hot_tensors[key] for key in example_keys]

    # get the images
    image_tensors = [train_image_tensors[key] for key in example_keys]

    # Training step (forward pass + backwards pass)
    total_loss = train_step_fn(image_tensors,
                               gt_boxes_list,
                               gt_classes_list,
                               detection_model,
                               optimizer,
                               to_fine_tune
                              )

    if idx % 10 == 0:
        print('batch ' + str(idx) + ' of ' + str(num_batches)
        + ', loss=' +  str(total_loss.numpy()), flush=True)

print('Done fine-tuning!')
```

```
Start fine-tuning!
batch 0 of 100, loss=1.1841527
batch 10 of 100, loss=22.973246
batch 20 of 100, loss=15.385794
batch 30 of 100, loss=3.1032684
batch 40 of 100, loss=0.39988923
batch 50 of 100, loss=0.08514348
batch 60 of 100, loss=0.035310157
batch 70 of 100, loss=0.000843467
batch 80 of 100, loss=0.0007025956
batch 90 of 100, loss=0.0006292969
Done fine-tuning!
```

Expected Output:

Total loss should be decreasing and should be less than 1 after fine tuning. For example:

```
Start fine-tuning!
batch 0 of 100, loss=1.2559178
batch 10 of 100, loss=16.067217
batch 20 of 100, loss=8.094654
batch 30 of 100, loss=0.34514275
batch 40 of 100, loss=0.033170983
batch 50 of 100, loss=0.0024622646
batch 60 of 100, loss=0.00074224477
batch 70 of 100, loss=0.0006149876
batch 80 of 100, loss=0.00046916265
batch 90 of 100, loss=0.0004159231
Done fine-tuning!
```

Load test images and run inference with new model!

You can now test your model on a new set of images. The cell below downloads 237 images of a walking zombie and stores them in a `results/` directory.

```
# uncomment if you want to delete existing files
!rm zombie-walk-frames.zip
!rm -rf ./zombie-walk
!rm -rf ./results

# download test images
!wget --no-check-certificate \
    https://storage.googleapis.com/laurencemoroney-blog.appspot.com/zombie-walk-frames.zip \
    -O zombie-walk-frames.zip

# unzip test images
local_zip = './zombie-walk-frames.zip'
zip_ref = zipfile.ZipFile(local_zip, 'r')
zip_ref.extractall('./results')
zip_ref.close()
```

```
rm: cannot remove 'zombie-walk-frames.zip': No such file or directory
--2021-05-22 23:53:22--  https://storage.googleapis.com/laurencemoroney-blog.appspot.com/zombi
Resolving storage.googleapis.com (storage.googleapis.com)... 142.250.101.128, 142.250.141.128,
Connecting to storage.googleapis.com (storage.googleapis.com)|142.250.101.128|:443... connecte
HTTP request sent, awaiting response... 200 OK
Length: 94778747 (90M) [application/zip]
Saving to: 'zombie-walk-frames.zip'

zombie-walk-frames. 100%[===================>]  90.39M   265MB/s    in 0.3s

2021-05-22 23:53:22 (265 MB/s) - 'zombie-walk-frames.zip' saved [94778747/94778747]
```

You will load these images into numpy arrays to prepare it for inference.

```
test_image_dir = './results/'
```

```
test_images_np = []

# load images into a numpy array. this will take a few minutes to complete.
for i in range(0, 237):
    image_path = os.path.join(test_image_dir, 'zombie-walk' + "{0:04}".format(i) + '.jpg')
    print(image_path)
    test_images_np.append(np.expand_dims(
      load_image_into_numpy_array(image_path), axis=0))
```

```
./results/zombie-walk0036.jpg
./results/zombie-walk0037.jpg
./results/zombie-walk0038.jpg
./results/zombie-walk0039.jpg
./results/zombie-walk0040.jpg
./results/zombie-walk0041.jpg
./results/zombie-walk0042.jpg
./results/zombie-walk0043.jpg
./results/zombie-walk0044.jpg
./results/zombie-walk0045.jpg
./results/zombie-walk0046.jpg
./results/zombie-walk0047.jpg
./results/zombie-walk0048.jpg
./results/zombie-walk0049.jpg
./results/zombie-walk0050.jpg
./results/zombie-walk0051.jpg
./results/zombie-walk0052.jpg
./results/zombie-walk0053.jpg
./results/zombie-walk0054.jpg

./results/zombie-walk0055.jpg
./results/zombie-walk0056.jpg
./results/zombie-walk0057.jpg
./results/zombie-walk0058.jpg
./results/zombie-walk0059.jpg
./results/zombie-walk0060.jpg
./results/zombie-walk0061.jpg
./results/zombie-walk0062.jpg
./results/zombie-walk0063.jpg
./results/zombie-walk0064.jpg
./results/zombie-walk0065.jpg
./results/zombie-walk0066.jpg
./results/zombie-walk0067.jpg
./results/zombie-walk0068.jpg
./results/zombie-walk0069.jpg
./results/zombie-walk0070.jpg
./results/zombie-walk0071.jpg
./results/zombie-walk0072.jpg
./results/zombie-walk0073.jpg
./results/zombie-walk0074.jpg
./results/zombie-walk0075.jpg
./results/zombie-walk0076.jpg
./results/zombie-walk0077.jpg
./results/zombie-walk0078.jpg
./results/zombie-walk0079.jpg
./results/zombie-walk0080.jpg
./results/zombie-walk0081.jpg
./results/zombie-walk0082.jpg
./results/zombie-walk0083.jpg
./results/zombie-walk0084.jpg
./results/zombie-walk0085.jpg
./results/zombie-walk0086.jpg
./results/zombie-walk0087.jpg
./results/zombie-walk0088.jpg
./results/zombie-walk0089.jpg
./results/zombie-walk0090.jpg
./results/zombie-walk0091.jpg
```

```
./results/zombie-walk0092.jpg
./results/zombie-walk0093.jpg
./results/zombie-walk0094.jpg
```

Exercise 11: Preprocess, predict, and post process an image

Define a function that returns the detection boxes, classes, and scores.

```
# Again, uncomment this decorator if you want to run inference eagerly
@tf.function
def detect(input_tensor):
    """Run detection on an input image.

    Args:
    input_tensor: A [1, height, width, 3] Tensor of type tf.float32.
      Note that height and width can be anything since the image will be
      immediately resized according to the needs of the model within this
      function.

    Returns:
    A dict containing 3 Tensors (`detection_boxes`, `detection_classes`,
      and `detection_scores`).
    """
    preprocessed_image, shapes = detection_model.preprocess(input_tensor)
    prediction_dict = detection_model.predict(preprocessed_image, shapes)

    ### START CODE HERE (Replace instances of `None` with your code) ###
    # use the detection model's postprocess() method to get the the final detections
    detections = detection_model.postprocess(prediction_dict, shapes)
    ### END CODE HERE ###

    return detections
```

You can now loop through the test images and get the detection scores and bounding boxes to overlay in the original image. We will save each result in a `results` dictionary and the autograder will use this to evaluate your results.

```
# Note that the first frame will trigger tracing of the tf.function, which will
# take some time, after which inference should be fast.

label_id_offset = 1
results = {'boxes': [], 'scores': []}

for i in range(len(test_images_np)):
    input_tensor = tf.convert_to_tensor(test_images_np[i], dtype=tf.float32)
    detections = detect(input_tensor)
    plot_detections(
      test_images_np[i][0],
      detections['detection_boxes'][0].numpy(),
      detections['detection_classes'][0].numpy().astype(np.uint32)
      + label_id_offset,
      detections['detection_scores'][0].numpy(),
      category_index, figsize=(15, 20), image_name="./results/gif_frame_" + ('%03d' % i) + ".jpg")
    results['boxes'].append(detections['detection_boxes'][0][0].numpy())
    results['scores'].append(detections['detection_scores'][0][0].numpy())
```

```
# TEST CODE

print(len(results['boxes']))
print(results['boxes'][0].shape)
print()

# compare with expected bounding boxes
print(np.allclose(results['boxes'][0], [0.28838485, 0.06830047, 0.7213766 , 0.19833465], rtol=0.18))
print(np.allclose(results['boxes'][5], [0.29168868, 0.07529271, 0.72504973, 0.20099735], rtol=0.18))
print(np.allclose(results['boxes'][10], [0.29548776, 0.07994056, 0.7238164 , 0.20778716], rtol=0.18
```

```
    237
    (4,)

    True
    True
    True
```

Expected Output: Ideally the three boolean values at the bottom should be `True`. But if you only get two, you can still try submitting. This compares your resulting bounding boxes for each zombie image to some preloaded coordinates (i.e. the hardcoded values in the test cell above). Depending on how you annotated the training images,it's possible that some of your results differ for these three frames but still get good results overall when all images are examined by the grader. If two or all are False, please try annotating the images again with a tighter bounding box or use the predefined `gt_boxes` list.

```
237
(4,)

True
True
True
```

You can also check if the model detects a zombie class in the images by examining the `scores` key of the `results` dictionary. You should get higher than 88.0 here.

```
x = np.array(results['scores'])

# percent of frames where a zombie is detected
zombie_detected = (np.where(x > 0.9, 1, 0).sum())/237*100
print(zombie_detected)
```

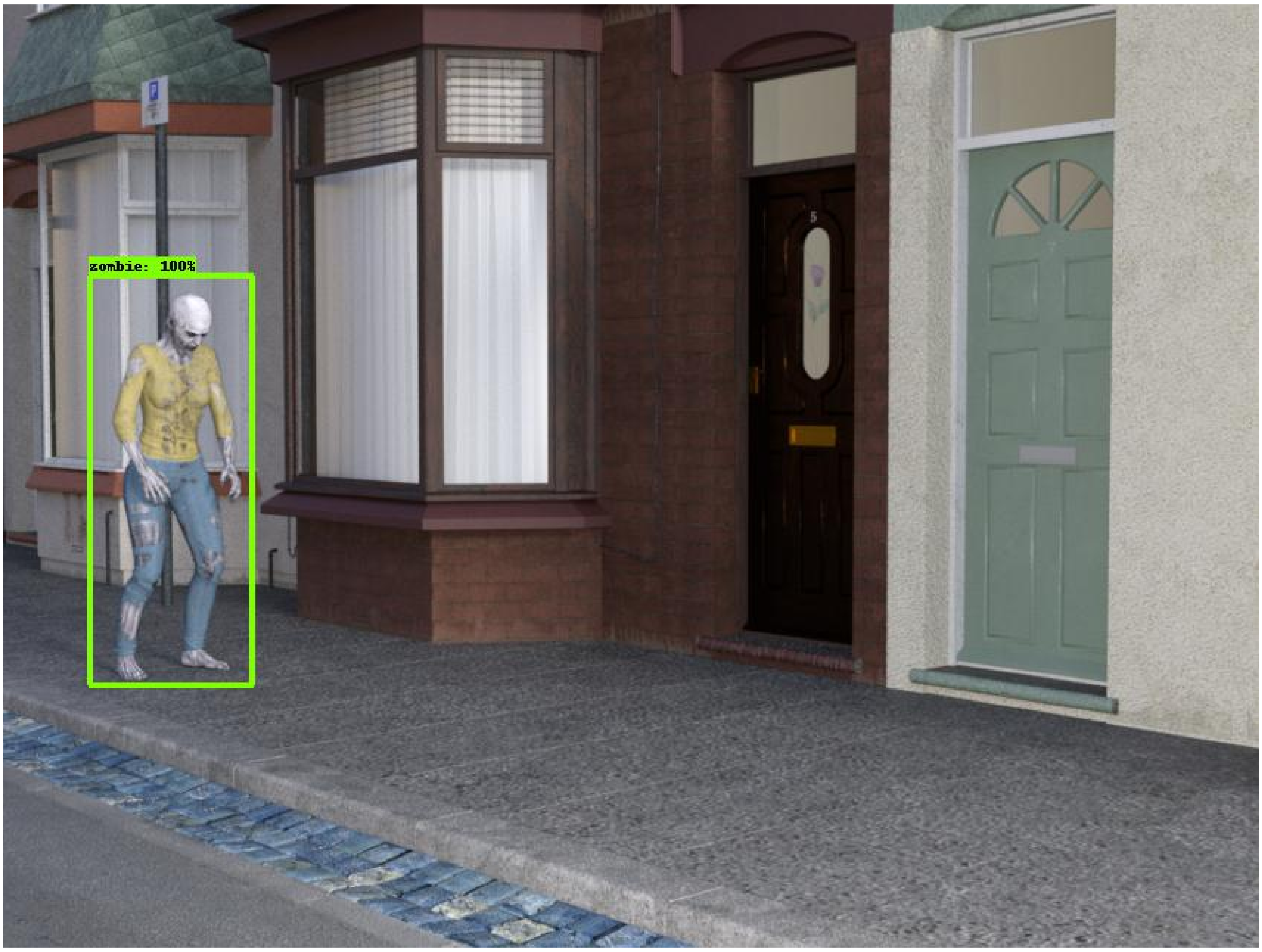
```
    97.0464135021097
```

You can also display some still frames and inspect visually. If you don't see a bounding box around the zombie, please consider re-annotating the ground truth or use the predefined `gt_boxes` here

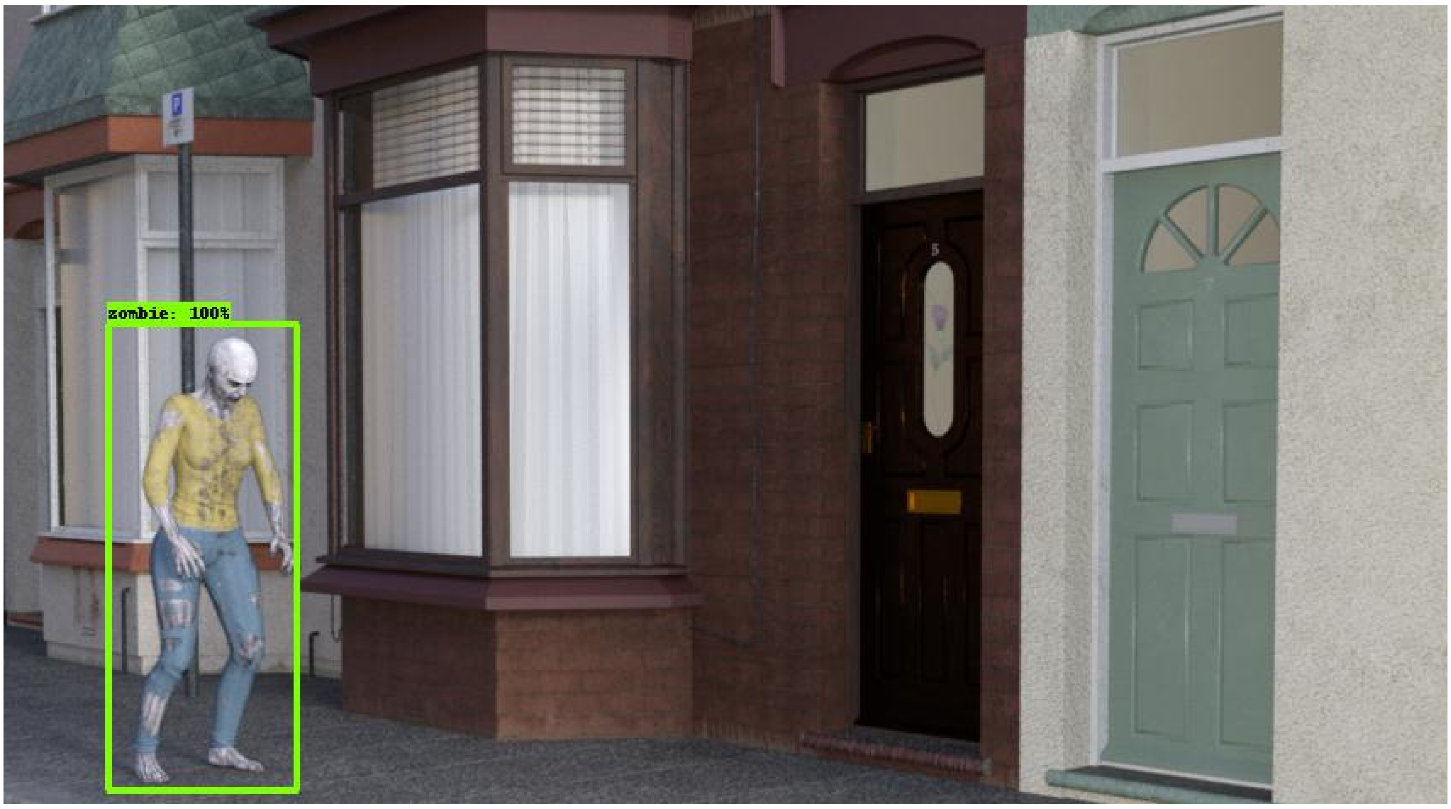
```
print('Frame 0')
display(IPyImage('./results/gif_frame_000.jpg'))
print()
print('Frame 5')
```

```
display(IPyImage('./results/gif_frame_005.jpg'))
print()
print('Frame 10')
display(IPyImage('./results/gif_frame_010.jpg'))
```

Frame 0



Frame 5



Create a zip of the zombie-walk images.

You can download this if you like to create your own animations

```
zipf = zipfile.ZipFile('./zombie-walk.zip', 'w', zipfile.ZIP_DEFLATED)
zipf = zipfile.ZipFile('./zombie.zip', 'w', zipfile.ZIP_DEFLATED)

filenames = glob.glob('./results/gif_frame_*.jpg')
filenames = sorted(filenames)

for filename in filenames:
    zipf.write(filename)

zipf.close()
```

Create Zombie animation

```
imageio.plugins.freeimage.download()

!rm -rf ./results/zombie-anim.gif

anim_file = './zombie-anim.gif'

filenames = glob.glob('./results/gif_frame_*.jpg')
filenames = sorted(filenames)
last = -1
images = []

for filename in filenames:
    image = imageio.imread(filename)
    images.append(image)

imageio.mimsave(anim_file, images, 'GIF-FI', fps=10)
```

Unfortunately, using `IPyImage` in the notebook (as you've done in the rubber ducky detection tutorial) for the large `gif` generated will disconnect the runtime. To view the animation, you can instead use the `Files` pane on the left and double-click on `zombie-anim.gif`. That will open a preview page on the right. It will take 2 to 3 minutes to load and see the walking zombie.

Save results file for grading

Run the cell below to save your results. Download the `results.data` file and upload it to the grader in the classroom.

```
import pickle

# remove file if it exists
!rm results.data

# write results to binary file. upload for grading.
with open('results.data', 'wb') as filehandle:
    pickle.dump(results['boxes'], filehandle)

print('Done saving! Please download `results.data` from the Files tab\n' \
      'on the left and submit for grading.\nYou can also use the next cell as a shortcut for downloa
```