Western_Detection

February 26, 2018

1 Western Detection

This notebook demonstrates Western Blot detection by an Inception network originally trained for Object Detection on the COCO Dataset, which was re-trained using Transfer Learning to identify Western blots.

For those who are unfamiliar, Western blotting is an approach for separating proteins by size, and images of Western blots in gels look like this:

Let's see how our model does in identifying the individual gels within that image.

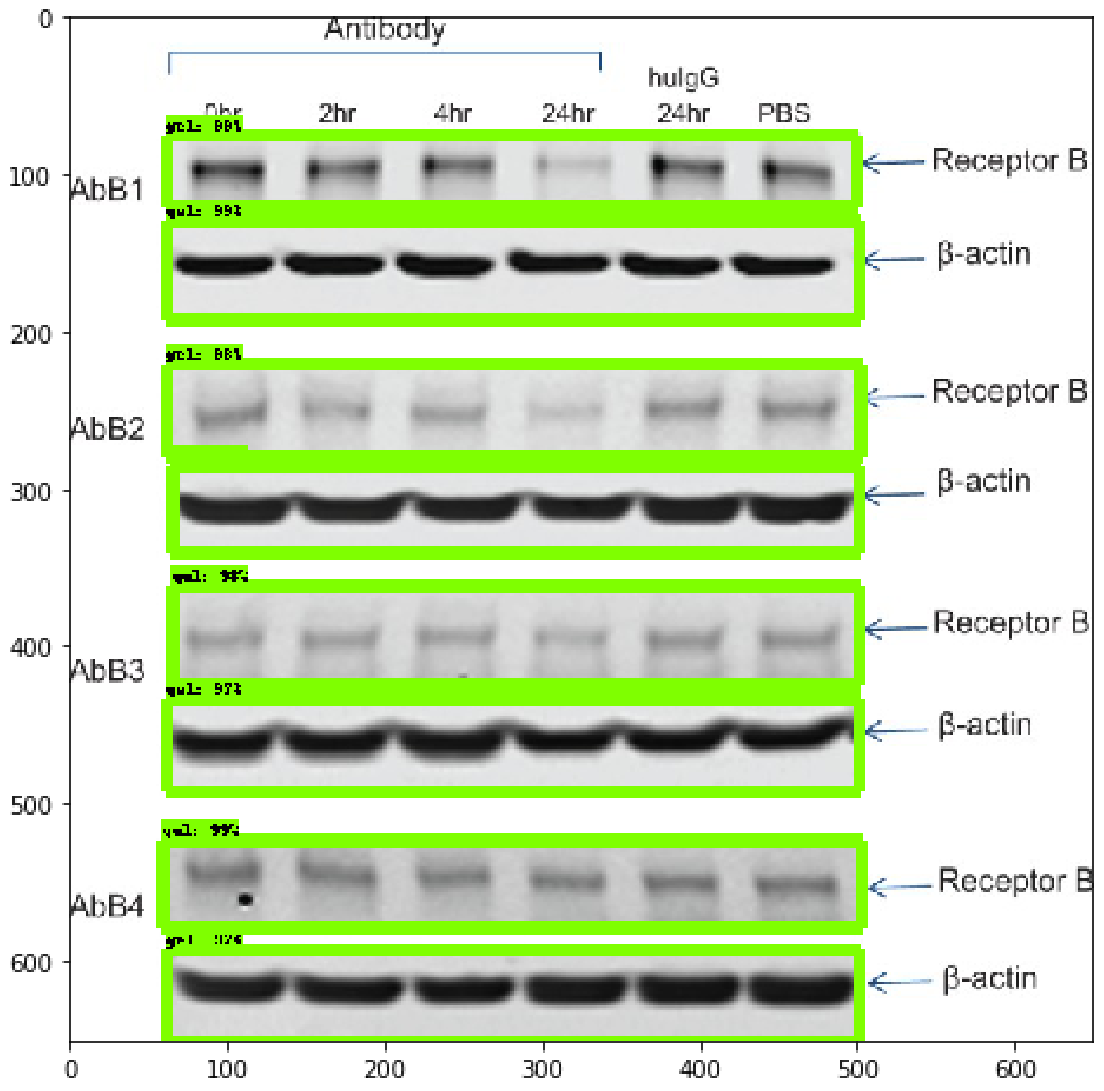
```
In [8]: # read in the dependencies
        import os
        import numpy as np
        from skimage import io
        from PIL import Image
        import base64
        import io
        import json
        import tensorflow as tf
        import sys
        import matplotlib
        %matplotlib inline
        from matplotlib import pyplot as plt
        # load in object detection utils dependencies
        sys.path.append('/Users/nweir/.virtualenvs/tensorflow/lib/python3.6/site-packages/tensor
        from utils import label_map_util
        from utils import visualization_utils as vis_util
```

```
In [4]: # load in the tensorflow model. The following is loosely based on the Google Object Dete
        # https://github.com/tensorflow/models/blob/master/research/object_detection/object_dete

        detection_graph = tf.Graph()
        with detection_graph.as_default():
          od_graph_def = tf.GraphDef()
          with tf.gfile.GFile('../exported_model/frozen_inference_graph.pb', 'rb') as fid:
            serialized_graph = fid.read()
            od_graph_def.ParseFromString(serialized_graph)
            tf.import_graph_def(od_graph_def, name='')
```

```
# load in label map
label_map = label_map_util.load_labelmap('../gel_label_map.pbtxt')
categories = label_map_util.convert_label_map_to_categories(label_map, max_num_classes=1
category_index = label_map_util.create_category_index(categories)

# set up inference
def infer(graph, image_path, category_index, min_score, output):
    with graph.as_default():
        with tf.Session(graph=graph) as sess:
            image_np = load_image_into_numpy_array(Image.open(image_path).convert('RGB')
            height, width, _ = image_np.shape
            image_np_expanded = np.expand_dims(image_np, axis=0)
            image_tensor = graph.get_tensor_by_name('image_tensor:0')
            boxes = graph.get_tensor_by_name('detection_boxes:0')
            scores = graph.get_tensor_by_name('detection_scores:0')
            classes = graph.get_tensor_by_name('detection_classes:0')
            num_detections = graph.get_tensor_by_name('num_detections:0')
            (boxes, scores, classes, num_detections) = sess.run(
                [boxes, scores, classes, num_detections],
                feed_dict={image_tensor: image_np_expanded})
            boxes = np.squeeze(boxes)
            classes = np.squeeze(classes).astype(np.int32)
            scores = np.squeeze(scores)
    for i in range(len(boxes)):
        confidence = float(scores[i])
        if confidence >= min_score:
            ymin, xmin, ymax, xmax = tuple(boxes[i].tolist())
            ymin = int(ymin * height)
            ymax = int(ymax * height)
            xmin = int(xmin * width)
            xmax = int(xmax * width)
            class_name = category_index[classes[i]]['name']
            output.append(
                {
                    'coordinates': {
                                    'y0': ymin,
                                    'y1': ymax,
                                    'x0': xmin,
                                    'x1': xmax
                                },
                    'label': class_name,
                    'confidence': confidence
                }
            )

# image pre-processing function
def load_image_into_numpy_array(image):
  (im_width, im_height) = image.size
```

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

    os.chdir('files/')
```

```
In [9]: # perform detection and visualize.
        IMAGE_SIZE = (12, 8)
        image_path = 'sample_western.jpg'
        with detection_graph.as_default():
          with tf.Session(graph=detection_graph) as sess:
            # Definite input and output Tensors for detection_graph
            image_tensor = detection_graph.get_tensor_by_name('image_tensor:0')
            # Each box represents a part of the image where a particular object was detected.
            detection_boxes = detection_graph.get_tensor_by_name('detection_boxes:0')
            # Each score represent how level of confidence for each of the objects.
            # Score is shown on the result image, together with the class label.
            detection_scores = detection_graph.get_tensor_by_name('detection_scores:0')
            detection_classes = detection_graph.get_tensor_by_name('detection_classes:0')
            num_detections = detection_graph.get_tensor_by_name('num_detections:0')
            image = Image.open(image_path)
            image = image.convert('RGB')
            # the array based representation of the image will be used later in order to prepare
            # result image with boxes and labels on it.
            image_np = load_image_into_numpy_array(image)
            # Expand dimensions since the model expects images to have shape: [1, None, None, 3]
            image_np_expanded = np.expand_dims(image_np, axis=0)
            # Actual detection.
            (boxes, scores, classes, num) = sess.run(
              [detection_boxes, detection_scores, detection_classes, num_detections],
              feed_dict={image_tensor: image_np_expanded})
            # Visualization of the results of a detection.
            vis_util.visualize_boxes_and_labels_on_image_array(
              image_np,
              np.squeeze(boxes),
              np.squeeze(classes).astype(np.int32),
              np.squeeze(scores),
              category_index,
              use_normalized_coordinates=True,
              line_thickness=8)
            plt.figure(figsize=IMAGE_SIZE)
            plt.imshow(image_np)
```



After detection as shown above, gel-containing boxes were extracted from 38,800 figures to produce the library of Western blot images described.