CS512 - PROJECT REPORT

Rescoring the classified Objects and Identification

By

Vidya Sudharshana(A20472468)
Sukanta Sharma(A20472623)

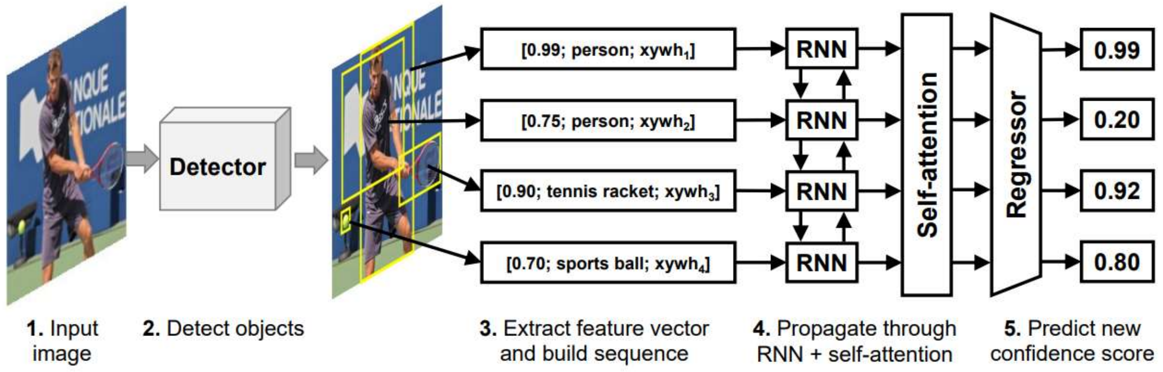
Problem Statement:

Objective of the exercise is to reevaluate the features detected by an existing feature detector and further improve detected objects by reducing the confidence of duplicate detection and removing out of context objects by verifying them against ground truths. Post processing of the predicted feature vectors of an arbitrary detector is done by using bidirectional RNN model with self-attention contextual rescoring. With the reference to the following research paper, we try to achieve the objective.

https://openaccess.thecvf.com/content_CVPR_2020/papers/Pato_Seeing_without_Looking_Contextual_Rescoring_of_Object_Detections_for_AP_CVPR_2020_paper.pdf

Approach:

Images from COCO dataset is fed into the existing detector models. Output features vectors are collected or stored in json format which is compatible with COCO data format. These feature vectors are fed into the model for rescoring and confidence evaluation. Post processing model includes cross checking the detected features against the annotations of the COCO dataset images to compare against the ground truths of the features and use as the target feature for the neural network model. Model includes a bidirectional RNN followed by a self-attention and a regressor unit. Model is implemented using TensorFlow/Keras.


Detector
[0.99; person; xywh$_1$]
[0.75; person; xywh$_2$]
[0.90; tennis racket; xywh$_3$]
[0.70; sports ball; xywh$_4$]
RNN
RNN
RNN
RNN
Self-attention
Regressor
0.99
0.20
0.92
0.80
1. Input image
2. Detect objects
3. Extract feature vector and build sequence
4. Propagate through RNN + self-attention
5. Predict new confidence score

Implementation:

1. COCO dataset images and annotations are downloaded from the following link
   a. http://images.cocodataset.org/zips/train2017.zip
   b. http://images.cocodataset.org/zips/val2017.zip
   c. http://images.cocodataset.org/annotations/annotations_trainval2017.zip
   d. http://images.cocodataset.org/zips/test2017.zip

2. ssd_mobilenet_v1_coco_2017_11_17 model is used to detect the feature vector of the images in training data set. Supporting environment for detecting the objects can be downloaded using the following command
   a. git clone --depth 1 https://github.com/tensorflow/models

3. Initialize the model detector

```
model_name = 'ssd_mobilenet_v1_coco_2017_11_17'
detection_model = load_model(model_name)
```

   In this exercise we are using SSD mobilenet model. Other models that can be used. load_model downloaded the corresponding model from the internet if its not already present.

4. Pass the images from the COCO dataset to the initialized model

```
      img = np.array(Image.open(i))
      predictions = run_inference_for_single_image(detection_model, img)
```

5. predictions from the model will have 4 dict keys which are as follows
   a. "detection_classes" – Contains the classes detected of the objects like person, motocycle, dog and so on
   b. "detection_boxes" – Contains the coordinates of the corners of the bounding boxes for the detected classes
   c. "detection_scores" – Contains of confidence scores of the detected features
   d. "category_id" -- Category ID is used to map the features to classes

   In most of the cases, length of the predictions vector is 100

6. Above predictions can be stored in json format following COCO format and later json file can be fed into the rescorer model or directly fed into the model. In this program we directly feed the output predictions into the rescorer model.

```
      stat = {"category_id":int(predictions['detection_classes'][i])
              "bbox":predictions['detection_boxes'][i].tolist(),
              "score":float(predictions['detection_scores'][i]),
             }
      pred_json.append(stat)
```

7. Predicted features need to be cross verified with the actual images and its features for getting the ground truth. COCO dataset annotations will be used to verify the features and get to know the ground truth of the feature. Load the annotations from the COCO dataset

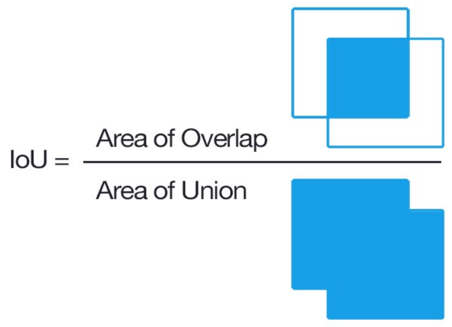
```
f_train = open(in_files_train_ann, 'r')
data = json.load(f_train)
categories = data['categories']
super_cats = {cat["id"]: cat["supercategory"] for cat in categories}
categories = {cat["id"]: cat for cat in categories}
category_index = list(categories.keys())
```

8. Preparing the input tensor for the rescorer model. Input tensor will have 85 features from the detector model. First feature will be the scores followed by category classes of 80 columns and 4 features of (xi / W, yi / H, wi / W, hi / H). Category classes are integer varying from 1 to 80 and made suitable using one hot encoder. W and H are the width and height of the image processed.

$$\mathbf{x}_i = [score_i] \oplus [\text{one\_hot}\,(class_i)] \oplus \left[\frac{x_i}{W}, \frac{y_i}{H}, \frac{w_i}{W}, \frac{h_i}{H}\right]$$

```
x, y, w, h = predictions['detection_boxes'][i]
bb_params = np.array([x/W, y/H, w/W, h/W], dtype=float)
scores = np.array(float(predictions['detection_scores'][i]), dtype=float)
clss = np.array(ohe(int(predictions['detection_classes'][i])),
                        dtype=float)
in_f = np.concatenate((scores,clss, bb_params), axis = None)
input_features = np.array(input_features)
in_features = tf.constant(input_features)
```

9. Preparing the target tensor feature
image_id, catergory_id, bbox fields from the extracted annotations are used to match the image from the dataset to the features from the detector mode. Intersection of union is calculated for the boundary boxes of the features. All the annotated features in which intersection of union is greater than 0.5 will be used as a target tensor to the rescorer model. IoU is calculated as below



IoU calculation:

```
def IoU(a, b):
    x1, y1, w1, h1 = a["bbox"]
    x2, y2, w2, h2 = b["bbox"]
    xA = max(x1, x2)
    yA = max(y1, y2)
    xB = min(w1, w2)
    yB = min(h1, h2)
    iA = max(0, xA - xB + 1) * max(0, yA - yB + 1)
    bxAA = (w1 - x1 + 1) * (h1 - y1 + 1)
    bxBA = (w2 - x2 + 1) * (h2 - y2 + 1)
    iou = iA / float(bxAA + bxBA - iA)
    return iou
```

10. Iterate over the list of predictions and annotations list to figure out the feature vectors for which IoU is greater than 50%. If no features are found in the annotation list with greater than 50% IoU, we treat the corresponding entry as a false positive for a feature. If features are present, then corresponding feature score from the predictions are used as an entry.

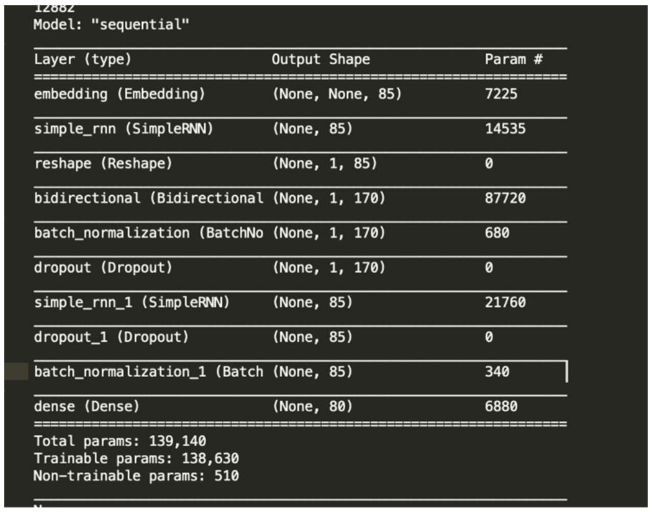
```
for i, pred in enumerate(pred_json):
    gts_class = []
    track_mat = []
    for j in range(len_ann):
        if(annotations[j]["category_id"] == pred["category_id"] and
                not cvrd[j] and IoU(pred, annotations[j]) >= threshold):
                #and annotations[j]["iscrowd"]):
            gts_class.append(annotations[j])
            track_mat.append(j)

    if len(gts_class) == 0:
        target_features_res.append(0)
        continue

    class_ious = IoU_matrix([pred], gts_class)
    matched_gt_class = class_ious.argmax()
    matched_gt = track_mat[matched_gt_class]
    cvrd[matched_gt] = True
    target_features_res.append(pred["score"])

target_features = tf.constant(target_features_res)
tf.reshape(target_features, [-1, 1])
```

11. With the above input and output feature tensors, a bidirectional RNN model based on GRU layer is built. Model has an embedding layer with 85 layers, 85 hidden layers, a bidirectional GRU layer, Batch normalization layers, dropout layers with dropping ratio of 30% and finally a dense layer with 80 outputs corresponding to the 80 encoded classes. Feature with the maximum value will be the class detected for that entry. Following is the summary of the model



```
12882
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, None, 85)          7225
_________________________________________________________________
simple_rnn (SimpleRNN)       (None, 85)                14535
_________________________________________________________________
reshape (Reshape)            (None, 1, 85)             0
_________________________________________________________________
bidirectional (Bidirectional (None, 1, 170)            87720
_________________________________________________________________
batch_normalization (BatchNo (None, 1, 170)            680
_________________________________________________________________
dropout (Dropout)            (None, 1, 170)            0
_________________________________________________________________
simple_rnn_1 (SimpleRNN)     (None, 85)                21760
_________________________________________________________________
dropout_1 (Dropout)          (None, 85)                0
_________________________________________________________________
batch_normalization_1 (Batch (None, 85)                340
_________________________________________________________________
dense (Dense)                (None, 80)                6880
=================================================================
Total params: 139,140
Trainable params: 138,630
Non-trainable params: 510
_________________________________________________________________
```

Code:

```
#Model
early_stopping = EarlyStopping(min_delta = 0.001,
                                          patience = 20,
                                          restore_best_weights=True)
model = keras.Sequential()
emb = model.add(layers.Embedding(input_dim=85, output_dim=85))
model.add(layers.SimpleRNN(85, input_shape=in_features.shape[1:]))
model.add(layers.Reshape((1,85)))
model.add(layers.Bidirectional(layers.GRU(85, return_sequences=True)))
model.add(layers.BatchNormalization())
model.add(layers.Dropout(0.3))
model.add(layers.SimpleRNN(85))
model.add(layers.Dropout(0.3))
model.add(layers.BatchNormalization())
model.add(layers.Dense(80, activation='relu'))
```

12. Model is compiled with adam optimizer, mse as loss having accuracy matrics. 30 and 128 are set as the epochs and batch_size for the model.

```
model.compile(optimizer='Adam', loss="mse", metrics=["accuracy"])
hist = model.fit(in_features, target_features, epochs=30, batch_size=128)
```

13. Model output is fed into an self-attention unit. Here each element summarizes whole sequence into a context vector ci given by average of all hidden vectors weighted by alignment score

$$\mathbf{c}_i = \sum_{j=1}^{L} \alpha_{ij} \mathbf{h}_j,$$

L length of the sequence, hj is the hidden vector of the element aij alignment between I and j. Weights are computed by softmax over alignment scores

$$\alpha_{ij} = \frac{\exp(\text{score}(\mathbf{h}_i, \mathbf{h}_j))}{\sum_{k=1}^{L} \exp(\text{score}(\mathbf{h}_i, \mathbf{h}_k))},$$

Scoring function is measured as below

$$\text{score}(\mathbf{h}_i, \mathbf{h}_j) = \frac{\mathbf{h}_i^\top \mathbf{h}_j}{\sqrt{L}}.$$

14. Regressor unit:
Multi-layer perceptron MLP, is used to predict the rescored confidence of each detection. Input to the regressor is GRU hidden vector h and context vector c. ReLU activation is used to get a score between 0 and 1, with the output feature dimension of 80x1

15. Loss function:

Rescoring is calculated using the following squared error

$$\mathcal{L}(\mathbf{y}, \mathbf{y}^*) = \sum_{i=1}^{L} (\mathbf{y}_i - \mathbf{y}_i^*)^2,$$

y is rescored confidence, y* is the target sequence.

Data used for evaluation:

Existing detectors on MS COCO data set are used to generate detections on train2017 (118k images) for training, val2017 (5k images) for model validation, and test-dev2017 (20k images) for evaluation. Annotations for the corresponding images were 2017 Train/Val annotations [241MB]. The above data sets can be downloaded from

- https://cocodataset.org/#download

Links for the dataset from google drive

- https://drive.google.com/drive/folders/1UdNZn4lqk_wkDLz1JniekihwP3qIsGTL?usp=sharing

Steps to execute the program:

1. To download the Tensorflow detector model
   a. git clone --depth 1 https://github.com/tensorflow/models
   b. If error is encountered while insertion of string_int_label_map_pb proto module, place the file string_int_label_map_pb2.py from the link below in the folder model/research/object_detection/protos/
   https://github.com/datitran/object_detector_app/blob/master/object_detection/protos/string_int_label_map_pb2.py
   c. Kindly make sure that source file main_prog.py, detection_model.py are in the same directory as models downloaded.

2. Installation of prerequisite python models
   a. pip install -r model/official/requirements.txt

3. Executing the program
   a. main_prog.py <train images folder> <training images annotations> <test images folder> <test images annotations > or
   b. main_prog.py <train images folder> <training images annotations>

4. Training, validation samples are present in the following link
   a. https://drive.google.com/drive/folders/1UdNZn4lqk_wkDLz1JniekihwP3qIsGTL?usp=sharing

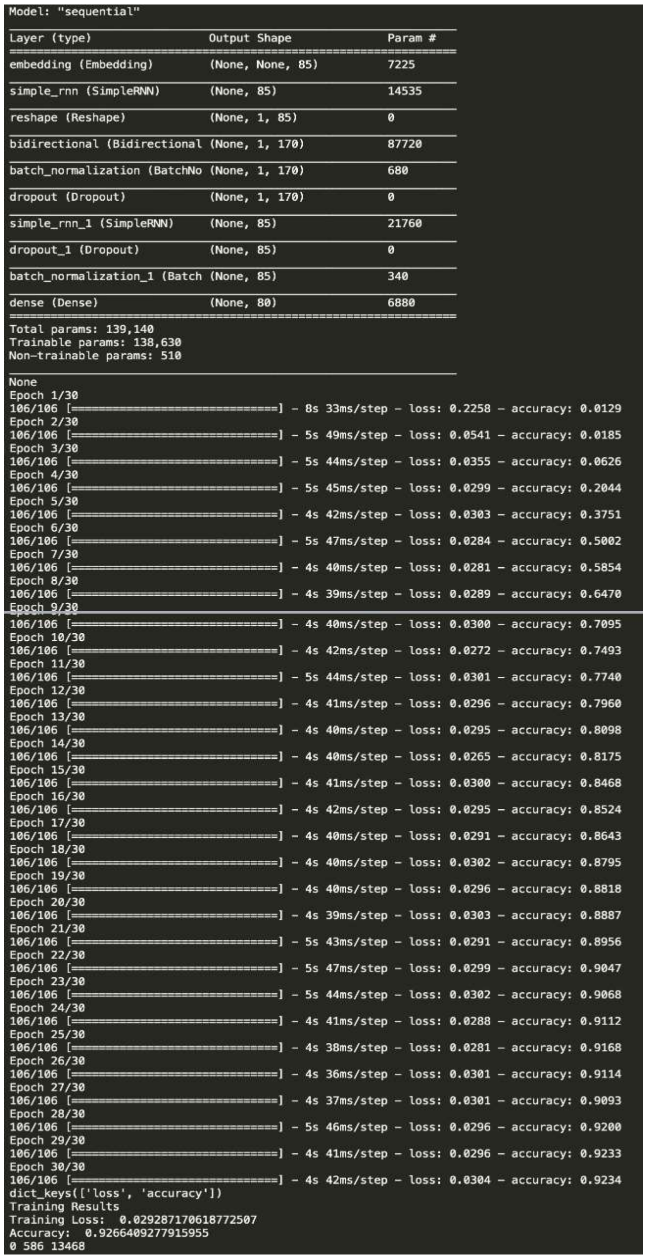
   Folder 5k – 5000 random samples from COCO dataset
   Folder 10K – 10000 random samples from COCO dataset
   Folder 15K – 15000 random samples from COCO dataset
   Folder val2017 – Validation samples from COCO dataset
   instances_train2017.json – Annotation for the training dataset
   instances_val2017.json – Annotation for the validation dataset

Experiment Results:

1. The above model, we were able to achieve 95% accuracy.
   In the example below, model removed 586 feature false detection from the detector model.
   Following are the loss, accuracy for training and validation datasets

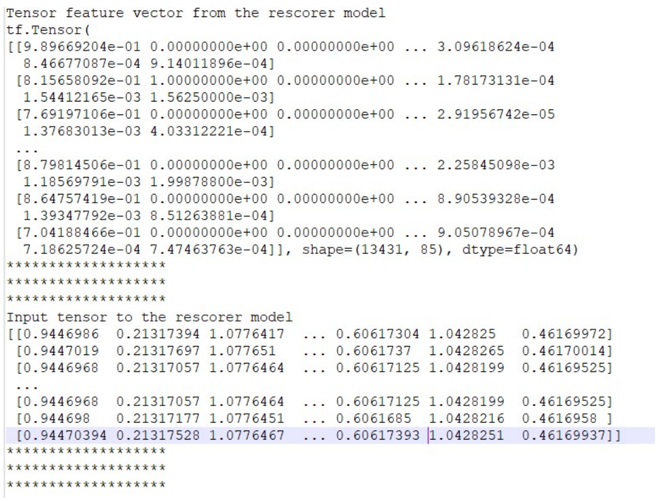
   Training Results
   Training Loss: 0.014138024300336838
   Accuracy: 0.9544382095336914

   Valuation Test results
   Valuation loss: 0.014138024300336838
   Valuation Accuracy: 0.9544382095336914



```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, None, 85)          7225
_________________________________________________________________
simple_rnn (SimpleRNN)       (None, 85)                14535
_________________________________________________________________
reshape (Reshape)            (None, 1, 85)             0
_________________________________________________________________
bidirectional (Bidirectional (None, 1, 170)            87720
_________________________________________________________________
batch_normalization (BatchNo (None, 1, 170)            680
_________________________________________________________________
dropout (Dropout)            (None, 1, 170)            0
_________________________________________________________________
simple_rnn_1 (SimpleRNN)     (None, 85)                21760
_________________________________________________________________
dropout_1 (Dropout)          (None, 85)                0
_________________________________________________________________
batch_normalization_1 (Batch (None, 85)                340
_________________________________________________________________
dense (Dense)                (None, 80)                6880
=================================================================
Total params: 139,140
Trainable params: 138,630
Non-trainable params: 510
_________________________________________________________________
None
Epoch 1/30
106/106 [==============================] - 8s 33ms/step - loss: 0.2258 - accuracy: 0.0129
Epoch 2/30
106/106 [==============================] - 5s 49ms/step - loss: 0.0541 - accuracy: 0.0185
Epoch 3/30
106/106 [==============================] - 5s 44ms/step - loss: 0.0355 - accuracy: 0.0626
Epoch 4/30
106/106 [==============================] - 5s 45ms/step - loss: 0.0299 - accuracy: 0.2044
Epoch 5/30
106/106 [==============================] - 4s 42ms/step - loss: 0.0303 - accuracy: 0.3751
Epoch 6/30
106/106 [==============================] - 5s 47ms/step - loss: 0.0284 - accuracy: 0.5002
Epoch 7/30
106/106 [==============================] - 4s 40ms/step - loss: 0.0281 - accuracy: 0.5854
Epoch 8/30
106/106 [==============================] - 4s 39ms/step - loss: 0.0289 - accuracy: 0.6470
Epoch 9/30
106/106 [==============================] - 4s 40ms/step - loss: 0.0300 - accuracy: 0.7095
Epoch 10/30
106/106 [==============================] - 4s 42ms/step - loss: 0.0272 - accuracy: 0.7493
Epoch 11/30
106/106 [==============================] - 5s 44ms/step - loss: 0.0301 - accuracy: 0.7740
Epoch 12/30
106/106 [==============================] - 4s 41ms/step - loss: 0.0296 - accuracy: 0.7960
Epoch 13/30
106/106 [==============================] - 4s 40ms/step - loss: 0.0295 - accuracy: 0.8098
Epoch 14/30
106/106 [==============================] - 4s 40ms/step - loss: 0.0265 - accuracy: 0.8175
Epoch 15/30
106/106 [==============================] - 4s 41ms/step - loss: 0.0300 - accuracy: 0.8468
Epoch 16/30
106/106 [==============================] - 4s 42ms/step - loss: 0.0295 - accuracy: 0.8524
Epoch 17/30
106/106 [==============================] - 4s 40ms/step - loss: 0.0291 - accuracy: 0.8643
Epoch 18/30
106/106 [==============================] - 4s 40ms/step - loss: 0.0302 - accuracy: 0.8795
Epoch 19/30
106/106 [==============================] - 4s 40ms/step - loss: 0.0296 - accuracy: 0.8818
Epoch 20/30
106/106 [==============================] - 4s 39ms/step - loss: 0.0303 - accuracy: 0.8887
Epoch 21/30
106/106 [==============================] - 5s 43ms/step - loss: 0.0291 - accuracy: 0.8956
Epoch 22/30
106/106 [==============================] - 5s 47ms/step - loss: 0.0299 - accuracy: 0.9047
Epoch 23/30
106/106 [==============================] - 5s 44ms/step - loss: 0.0302 - accuracy: 0.9068
Epoch 24/30
106/106 [==============================] - 4s 41ms/step - loss: 0.0288 - accuracy: 0.9112
Epoch 25/30
106/106 [==============================] - 4s 38ms/step - loss: 0.0281 - accuracy: 0.9168
Epoch 26/30
106/106 [==============================] - 4s 36ms/step - loss: 0.0301 - accuracy: 0.9114
Epoch 27/30
106/106 [==============================] - 4s 37ms/step - loss: 0.0301 - accuracy: 0.9093
Epoch 28/30
106/106 [==============================] - 5s 46ms/step - loss: 0.0296 - accuracy: 0.9200
Epoch 29/30
106/106 [==============================] - 4s 41ms/step - loss: 0.0296 - accuracy: 0.9233
Epoch 30/30
106/106 [==============================] - 4s 42ms/step - loss: 0.0304 - accuracy: 0.9234
dict_keys(['loss', 'accuracy'])
Training Results
Training Loss:  0.029287170618772507
Accuracy:  0.9266409277915955
0 586 13468
```

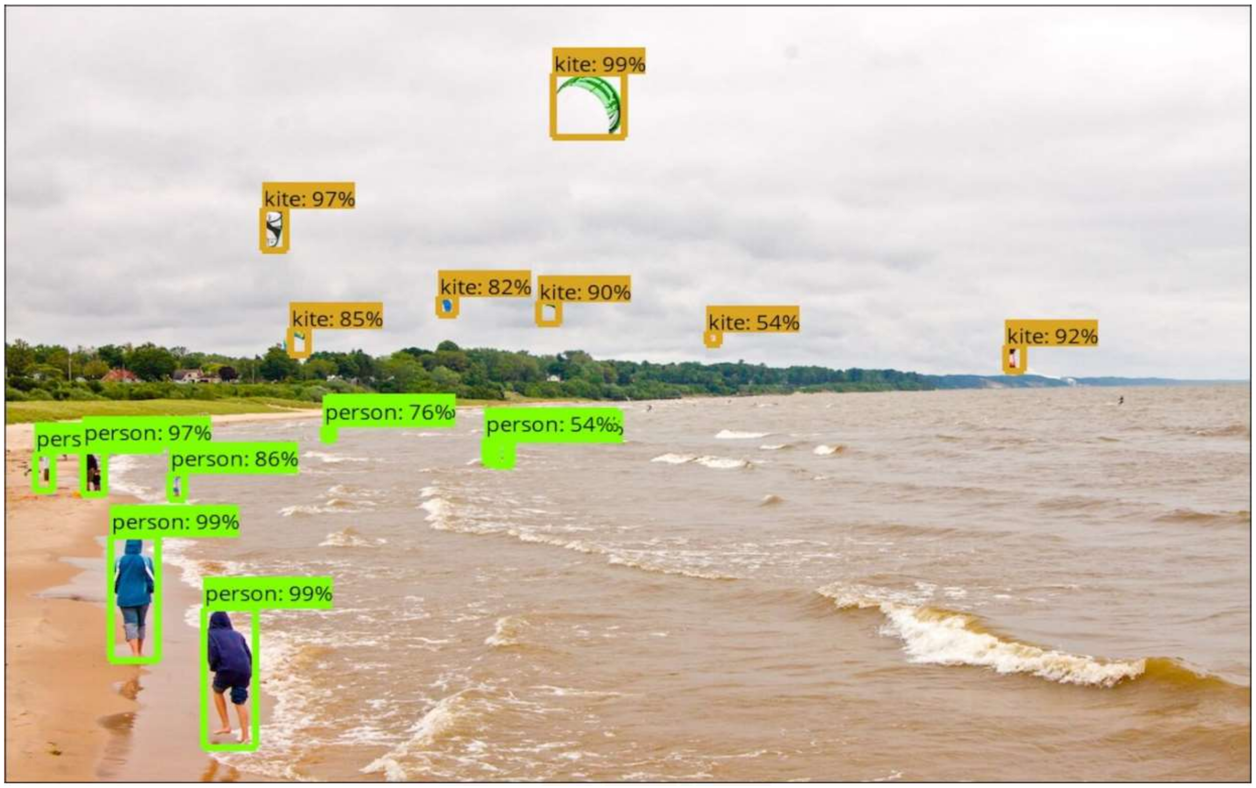
Input and output tensor vectors of the rescorer model:

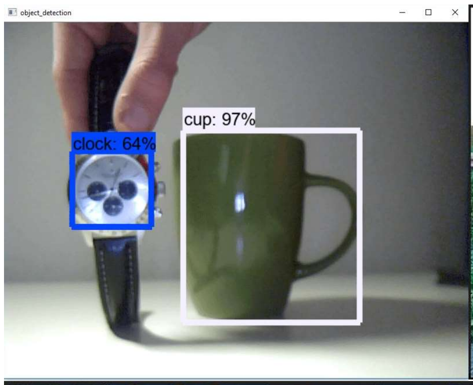


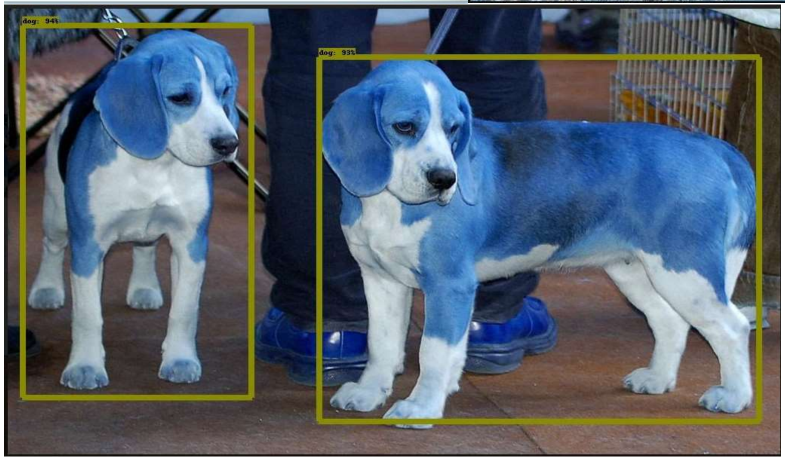
```
Tensor feature vector from the rescorer model
tf.Tensor(
[[9.89669204e-01 0.00000000e+00 0.00000000e+00 ... 3.09618624e-04
  8.46677087e-04 9.14011896e-04]
 [8.15658092e-01 1.00000000e+00 0.00000000e+00 ... 1.78173131e-04
  1.54412165e-03 1.56250000e-03]
 [7.69197106e-01 0.00000000e+00 0.00000000e+00 ... 2.91956742e-05
  1.37683013e-03 4.03312221e-04]
 ...
 [8.79814506e-01 0.00000000e+00 0.00000000e+00 ... 2.25845098e-03
  1.18569791e-03 1.99878800e-03]
 [8.64757419e-01 0.00000000e+00 0.00000000e+00 ... 8.90539328e-04
  1.39347792e-03 8.51263881e-04]
 [7.04188466e-01 0.00000000e+00 0.00000000e+00 ... 9.05078967e-04
  7.18625724e-04 7.47463763e-04]], shape=(13431, 85), dtype=float64)
*******************
*******************
*******************
Input tensor to the rescorer model
[[0.9446986  0.21317394 1.0776417  ... 0.60617304 1.042825   0.46169972]
 [0.9447019  0.21317697 1.077651   ... 0.6061737  1.0428265  0.46170014]
 [0.9446968  0.21317057 1.0776464  ... 0.60617125 1.0428199  0.46169525]
 ...
 [0.9446968  0.21317057 1.0776464  ... 0.60617125 1.0428199  0.46169525]
 [0.944698   0.21317177 1.0776451  ... 0.6061685  1.0428216  0.4616958 ]
 [0.94470394 0.21317528 1.0776467  ... 0.60617393 1.0428251  0.46169937]]
*******************
*******************
*******************
```

In the above example of input tensors and output, we can see the detector identified the first feature with 94% accuracy, which was a false positive and rescorer model reevaluated it to almost zero.

Rescored outputs:







Challenges:

1. Long evaluation time:
   a. COCO data is a very huge testing data set of approximately 21GB with over 100K images. Rescoring algorithm took a very long time for more than 12 hours without completion.
   b. To reduce the testing time, I tried to bring down the input feature vectors which needed to be iterated with annotations from the COCO. Here we only choose predictions with greater than 50 % and excluded other. Still we could not see training completion.

```
if (stat["score"] > 0.5):
   pred_json.append(stat)
   input_features.append(in_f)
```

   c. So, we reduced the data set to 3 different sizes, 5K, 10K and 15K. The number of feature vector with 5K testing samples were over 500K and for 15K samples 1500K respectively. If we consider all the prediction values. Again, we took predictions more than 50% and reduced the feature vectors to 13468 and 41063 respectively for 5K and 15K.
   d. Due to the reduced sample size, our accuracy was affected.

2. Padding sequence were resulting in target features to be filled with zeros for self-attention. This resulted in wrong evaluation of scores resulting in reduced accuracy. This was one of the affects which raised due to reduced sampling size. There were times when the feature vector scores were less and padding rounded off them to zero, if too high rounded them to 1. With zeros and 1's in the target sensor, model did not predict accurately with either 100% accuracy or 0%. Hence in the final model we had to take the attention layer out to achieve stable results.

Bibliography:

1. https://github.com/tensorflow/models
2. https://openaccess.thecvf.com/content_CVPR_2020/papers/Pato_Seeing_without_Looking_Contextual_Rescoring_of_Object_Detections_for_AP_CVPR_2020_paper.pdf
3. http://images.cocodataset.org/zips/train2017.zip
4. http://images.cocodataset.org/zips/val2017.zip
5. http://images.cocodataset.org/annotations/annotations_trainval2017.zip
6. http://images.cocodataset.org/zips/test2017.zip
7. https://towardsdatascience.com/master-the-coco-dataset-for-semantic-image-segmentation-part-1-of-2-732712631047
8. https://towardsdatascience.com/object-detection-simplified-e07aa3830954
9. https://towardsdatascience.com/object-detection-using-deep-learning-approaches-an-end-to-end-theoretical-perspective-4ca27eee8a9a
10. https://missinglink.ai/guides/tensorflow/building-faster-r-cnn-on-tensorflow-introduction-and-examples/
11. https://medium.com/object-detection-using-tensorflow-and-coco-pre/object-detection-using-tensorflow-and-coco-pre-trained-models-5d8386019a8
12. https://www.tensorflow.org/api_docs/python/tf/keras/layers/GRU
13. https://www.tensorflow.org/guide/keras/rnn
14. https://www.datacamp.com/community/news/deep-learning-rnn-lstm-gru-using-tensorflow-in-python-lzyhy9kef7b
15. https://www.tensorflow.org/api_docs/python/tf/keras/layers/Attention