Department of Computing

Imperial College of Science, Technology and Medicine

H-PERL: The Hybrid Property, Event and Relation Learner

Author:
Ross Irwin

Supervisors:
Prof. Alessandra Russo
Dr. Krysia Broda
Dr. Jorge Lobo

June 11, 2020

Chapter 1

Evaluation

This chapter describes the performance details of the components and models outlined in Chapters **??** and **??**. Section 1.1 compares the performance of a selection of counterpart components used in the hardcoded and trained models. Section 1.2 compares the overall performance of the two H-PERL models, and includes details of the models' performance under noisy conditions.

Before presenting the evaluation, we outline the following three key criteria that we want to evaluate the models against:

1. **Accuracy** :- What proportion of questions does the model answer correctly?
2. **Speed** :- How long does the model take to complete the evaluation?
3. **Adaptability** :- How simple is it to transfer the model to a new environment?

While we would have preferred to have compared our models to state-of-the-art neural network implementations for VideoQA, training end-to-end VideoQA networks is very resource intensive and takes a long time. If these models were included in the evaluation, we would add explainability as an additional criterion. However, since our models are both hybrid, their implementations have roughly the same explainability. We do, however, keep in mind that hybrid models are often easier to understand than fully-neural counterparts, as well as being significantly faster to train.

1.1 Components

The first part of the evaluation concerns the individual model components. In this section we compare the performance of the properties, relations and events components from both the hardcoded and trained models. These components are evaluated using perfect input data - where we assume no mistakes have been made in previous parts of the pipeline. We also outline the performance of the object detection component, which is common to both H-PERL models. All of these components are evaluated using the 200 validation videos, rather than the testing videos, although both sets of videos are generated in the same way.

This section does not include a dedicated discussion on the object tracker's performance, since there are no QA pairs with which it can be directly evaluated. However, after using some intuitive heuristics to gauge the component's object tracking abilities, we see that the tracker assigns identifiers in exactly the way we would expect for the OceanQA environment. In the rest of this chapter, therefore, it should be assumed that the tracker operates with perfect accuracy on the OceanQA environment, but we cannot provide an official evaluation for this.

1.1.1 Object Detector

In Chapter **??** we mentioned that *Intersection over Union* (IoU), which is calculated between a ground truth bounding box and the bounding box produced by the detector, measures how well a detector localises an image. As a reminder, the IoU between an object A and object B is defined as the area of intersection between A and B, divided by the area of union between A and B. Chapter **??** also mentioned that a threshold t can be applied to a set of detections so that only the detections with confidence $> t$ are used. Each IoU threshold that is applied to the set of detections produces a different precision-recall curve for the detector. The average precision (AP) is then defined as the area under the precision-recall curve, although estimates of this value are used. Average precision, however, is only defined for detection of a single class. The mean average precision (mAP) metric is therefore used to find the mean AP across k classes, and is intuitively defined as follows:

$$mAP = \frac{\sum_{i=1}^{k} AP_i}{k} \tag{1.1}$$

In order to conduct the evaluation, we used the *PyCocoTools*[1] library, which implements the metrics used in the COCO object detection challenge. The library provides mAP values for IoU thresholds 0.5 and 0.75, as well as the average mAP over the set of thresholds $\{0.5, 0.55, 0.6, ..., 0.95\}$. Table 1.1 presents mAP values achieved by our detector on the OceanQA validation dataset.

IoU Threshold	mAP
0.5	1.000
0.75	0.998
0.5 : 0.95	0.995

Table 1.1: Mean average precision values for a number of intersection over union thresholds. A threshold of 0.5 : 0.95 indicates an average mAP value for the set of thresholds: $\{0.5, 0.55, ..., 0.95\}$.

[1]Available at: https://github.com/cocodataset/cocoapi

The results presented in Table 1.1 show that our object detector performs very strongly. This isn't particularly surprising, since the dataset environment is very simple and the object detection algorithm used is state-of-the-art. Nonetheless it proves the assumption that it is possible, in the OceanQA environment at least, to extract objects from the frames of the video and work with these objects directly.

Despite the simpicity of the environment, the object detector is, however, quite slow. Detection of all objects in the testing dataset takes about 56 seconds; making the detector the slowest component in the trained model, and the second slowest in the hardcoded model. The full details of the timings of all components are outlined in TODO. Since the object detector can be trained for a specific environment, we consider it to be adaptable. Training can take a long time however, and each environment must have a large number of object detection examples available.

1.1.2 Properties

Since the properties component is predicting multiple properties for each object, the component has a separate accuracy for each property. As well as accuracy, we also evaluate the properties component in terms of precision and recall (all three metrics are defined in Chapter **??**), to ensure underrepresented property values are not being missed. Table 1.2 and Table 1.3 show the results for the hardcoded model on the *colour* and *rotation* properties, respectively. We do not present the results for the trained model, since every metric is 1.0 for all property values.

The results show that both the hardcoded and trained models have (almost) perfect performance. It is surprising that the trained model can achieve the same result as the hardcoded model, despite only using weakly-supervised QA training data - data where most objects are not labelled with property values. It is

Colour	**Accuracy**	**Precision**	**Recall**
red	1.000	1.000	1.000
blue	1.000	1.000	1.000
purple	1.000	1.000	1.000
brown	1.000	1.000	1.000
green	1.000	1.000	1.000
silver	1.000	1.000	1.000
white	1.000	1.000	1.000

Table 1.2: Hardcoded properties component results on the colour property.

Rotation	**Accuracy**	**Precision**	**Recall**
upward-facing	1.000	1.000	0.999
right-facing	1.000	0.999	1.000
downward-facing	1.000	0.999	1.000
left-facing	1.000	1.000	1.000

Table 1.3: Hardcoded properties component results on the rotation property.

worth noting, however, that this analysis uses the full-data version of the validation dataset; this means that the object detection was performed without any errors. However, even when the trained object detector was used, Section 1.2 shows that the properties questions were answered almost perfectly.

Despite being made of neural networks, the properties components are both fairly fast. Both components took approximately 18 seconds to label the entire all of the objects in the testing data with properties. As with the detector, both property component implementations are deemed adaptable, since they can simply be retrained for a new environment. One potential issue for the trained properties component, however, is a lack of data. The QA training data is sparsely labelled to start with, and, since training this component requires solving an optimisation problem, there may not be enough data for the optimiser to find a 'good' solution. However, we did not have time to explore this potential issue any further.

1.1.3 Relations

Since the task of the relations component is binary classification, and since the data is highly imbalanced we present the results for each implementation using a confusion matrix. A confusion matrix presents the true positives (TP), false postives (FP), true negatives (TN) and false negatives (FN) in a matrix. Unlike simply looking at the accuracy, confusion matrices are very useful for ensuring that the model is good at correctly predicting both positive and negative classes. The results for the hardcoded and trained relations components are shown in Table 1.4a and Table 1.4b, respectively.

Table 1.4a shows that the hardcoded relations component achieves an accuracy, precision and recall of 1.0, and therefore an F1-score of 1.0. The trained relations component performs marginally worse, with an accuracy of 0.99, a precision of 0.96, a recall of 1.0, and therefore an F1-score of 0.98. The performance of both components is, however, comparable.

While the hardcoded relations component takes only 22.8 seconds to process the entire testing dataset, the trained component takes almost twice as long, at 39.6 seconds. This time difference is due to the trained component using a neural function, rather than a hardcoded function. However, despite being slower, the trained component is more adaptable than the hardcoded component, since it can learn the definition of binary relations from the data.

Pred: / Actual:	Yes	No
Yes	0.16	0.00
No	0.00	0.84

(a) Hardcoded relations component.

Pred: / Actual:	Yes	No
Yes	0.16	0.00
No	0.01	0.83

(b) Trained relations component

Table 1.4: Confusion matrices for hardcoded and trained relations components for the *close* relation. Columns show predicted classifications, while rows show actual.

1.1.4 Events

1.2 Models