DOI: DOIunavailable

Software

- Review
- Repository
- Archive

Editor: Pending Editor

Reviewers:

- @Pending Reviewers

Submitted: N/A
Published: N/A

title: "Logodetect: One-shot detection of logos in image and video data"
tags:
- Python
- object detection
- object recognition
- computer vision
- image processing
- video processing
- one-shot learning
authors:
- name: Jorge Davila-Chacon
affiliation: 1
- name: Max Pumperla
affiliation: "2, 3"
affiliations:
- name: Heldenkombinat Technologies GmbH
index: 1
- name: IUBH Internationale Hochschule
index: 2
- name: Pathmind Inc.
index: 3
date: 13 August 2017
bibliography: paper.bib

Summary

`Logodetect` allows the detection of logos in images and videos after being trained with one example of the required logo. One-shot learning systems are not only great because they require little data, but also because they practically remove the need for specialists whenever the user wants to extend or re-purpose their functionality. This means that the benefits are manifold: the user requires little effort to collect and label training data, there is practically no time or economic costs for the training procedure, and it also provides a strategic benefit for business as they become self-sufficient for a large part of the system maintenance.

Statement of need

There is plenty of literature on the use of deep-learning for detecting logos, so, additionally to sharing with the community a couple of algorithms to get started with one-shot logo-detection, the aim of `Logodetect` is to provide a flexible architecture to facilitate the comparison of different algorithms for one-shot object recognition.

The inference pipeline of `Logodetect` supports architectures with one or two stages. **One-stage** architectures directly perform *object-recognition* on the raw images, and **two-stage** architectures first perform *object-detection* and then *object-recognition* in a second stage. The idea is that the user can use a generic detector for a single class of objects (e.g. logos, traffic signs or faces) and then compare each of its detections with the exemplar, i.e., the sub-class that the user is trying to recognize, to determine if both belong to the same sub-class (e.g. a concrete brand, a stop sign or the face of a loved one). To get started, we include two algorithms that the user can play with. Both have a Faster-RCNN (**NIPS2015_14bfa6bb?**) in the first stage that performs object-detection and they differ in the second stage that performs object-recognition.

Architecture

As a baseline, we bring the exemplars and the detections from the first stage to the same latent space (this reduces the course of dimensionality) and then simply measure the Euclidean or the Cosine distance between both embeddings for object-recognition. Both inputs are considered to belong to the same sub-class if their distance is below a threshold determined in a preliminary analysis of the training dataset. The code also provides functionality to add various transformations, so the user has the option to augment each exemplar with different transformations if desired. The user simply adds one or more exemplars into the `data/exemplars` folder that is generated after following the installation instructions below, and is good to go.

As a first reference against the baseline, we also provide a modified ResNet (**deep-residual-reco?**) for object-recognition that directly takes the exemplars and the detections from the first stage and predicts if both belong to the same sub-class. Similarly to (**NEURIPS2019_92af93f7?**), this network infers a distance metric after being trained with examples of different sub-classes, but instead of sharing the same weights and processing each input in a separate pass as in (**Koch2015SiameseNN?**), it concatenates both inputs and processes them in one pass. This concept follows more closely the architecture proposed by (**BHUNIA2019106965?**), where the assumption is that the exemplars often have more high-frequency components than the detections, and therefore the model can increase its accuracy by learning a separate set of weights for each input.

Acknowledgements

We would like to thank *NullConvergence* and all the contributors to the open-source framework `mtorch` (**mtorch?**) which serves as the foundation of our training pipeline.

References