Machine Learning Engineer Nanodegree

Capstone Project

Ann-Kristin Juschka

May 1, 2019

I. Definition

Project Overview

In this Nanodegree, *Convolutional Neural Networks (CNNs)* are introduced that are most commonly used to analyze visual data. In the deep learning project of the Nanodegree, we train CNNs to detect dogs in images and to classify dog breeds. In general, object detection and classification is a classical task in machine learning.

The aim of this project is to use CNNs to detect and classify brands in images with logos. For this, usually CNNs are trained with the dataset *FlickrLogos-27* [KPT+11] and *FlickrLogos-32* [RPLvZ11]; e.g. in [BBMS17, ISGK15]. We are also interested in this task as we later wish to *analyze sentiments* in twitter tweeds with pictures where a given company's logo is detected. Due to time and size limits, we focus in this project on *logo detection and classification.*

Problem Statement

While most logo datasets like *FlickrLogos-27* [KPT+11] and *FlickrLogos-32* [RPLvZ11] contain raw original logo graphics, we want to train our CNN on the in-the-wild logo dataset *Logos in the Wild* [THMB18] whose images contain logos as natural part. As this dataset includes in total 11,054 images with 32,850 annotated logo bounding boxes of 871 brands, it should be possible to train a CNN that achieves a high accuracy or mean average precision (map). This is a challenging task as the regions containg logos are small.

The main goal of this project is to use CNNs to *classify the brand and company logos* from the Logos in the Wild dataset with high accuracy and mean average precision (map). For this, we first train an own CNN architecture as done in the dog breeds project of the Nanodegree. Furthermore, to improve loss and accuracy we make use of the standard CNN architectures *VGG19, Resnet50, InceptionV3, or Xception* that are already trained on the *ImageNet* database [ker]. Moreover, as the dataset comes with annotations in Pascal-VOC style, if time permits we will also train a *Faster Region-based Convolutional Neural Network (Faster R-CNN)* [RHGS15] for two stages: First, an

Region Proposal Network (RPN) for *logo detection*, and second, a classifier like VGG19 for logo classification of the candidate regions. This Faster R-CNN will be evaluated using the *mean average precision (map)* metric.

Metrics

For purely logo classification, we seek to achieve a high accuracy. *Accuracy* is simply the number of correct predictions divided by the total number of predictions.

For logo detection, we need a different metric: *mean average precision (map)* that is the mean over all classes, of the interpolated *average precision* [EVGW+10] for each class. Recall that

$$\text{precision} = \frac{\text{true positives}}{\text{true positives} + \text{false positives}}, \quad \text{recall} = \frac{\text{true positives}}{\text{true positives} + \text{false negatives}}.$$

Considering the precision-recall curve for a given threshold, and the interpolated precision $\text{precision}_{\text{interpolated}}(\text{recall}_i) = \max_{\tilde{r}, \tilde{r} \geq \text{recall}_i} \text{precision}(\tilde{r})$ of the 11 values $\text{recall}_i = \{0, 0.1, 0.2, \ldots, 0.9, 1.0\}$. Then

$$\text{average precision} = \frac{1}{11} \sum_{1}^{11} \text{precision}_{\text{interpolated}}(\text{recall}_i) \qquad [\text{EVGW}^{+}10, \S\, 4.2].$$

II. Analysis

Data Exploration

To train our model, we want to use the recent logo dataset *Logos in the Wild* [THMB18]. As mentioned above, the lastest version of this dataset (v2.0) contains 11,054 images with 32,850 annotated logo bounding boxes of 871 brands and it is collected by performing Google image searches with well-known brand and company names directly or in combination with a predefined set of search terms like advertisement, building, poster or store. The logo annotations are in Pascal-VOC style.

As stated by the creators of the Logo in the Wild dataset, this dataset has 4 to 608 images per searched brand, 238 of 871 brands occur at least 10 times, and there are up to 118 logos in one image. Unfortunately, the dataset provides only the links to the images, and some of these images already disappeared. As we later want to detect logos in arbitrary pictures from twitter tweeds, this large in-the-wild logo dataset still fits best to our goal.

Instead of downloading ourselves the images from the different urls provided in the Logos in the Wild dataset, we download the *QMUL-OpenLogo Dataset* [SZG18], which contains Logos in the Wild as a subset including all available JPEG files.

We run a simple Python script to analyze the downloaded JPEG images in the Logos in the Wild dataset, and we find that in fact there are in total 821 brands, where the maximal number of logos per given brand is 1,928 for Heineken, and the minimum number is 1. Moreover, the image heineken/img00042.jpg contains the maximum of 118 logos, while in general every image contains at least one logo.

Exploratory Visualization

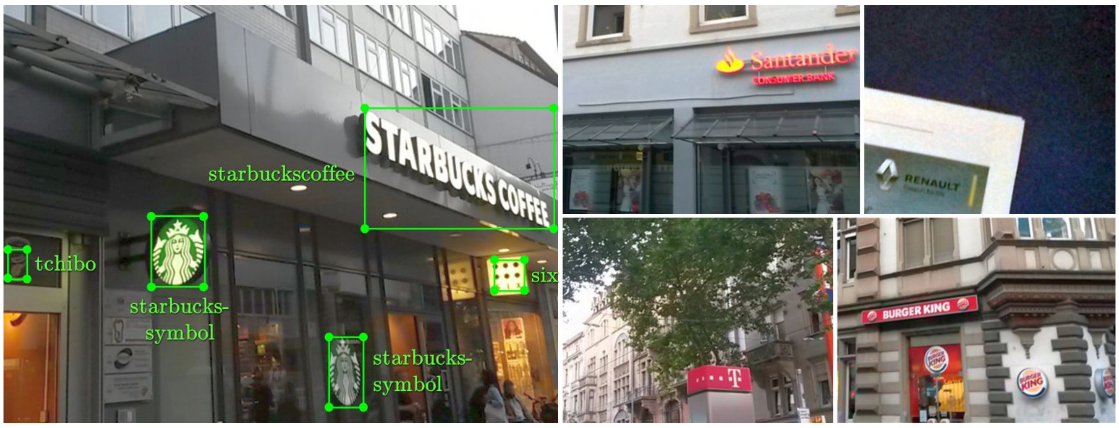

Figure 1: `https://www.iosb.fraunhofer.de/servlet/is/78045/`

While the images on the right contain logos of a single brand, the image on the left shows a sample image that includes bounding boxed for logos of different brands. In particular, an image can contain different types of logos of a brand. The following are the annotations in the Pascal-VOC style in the xml-file corresponding to the image on the left hand side:

```
<annotation>
  <folder>0samples</folder>
  <filename>img000009</filename>
  <path>\0samples\img000009.jpg</path>
  <source>
    <database>Unknown</database>
  </source>
  <size>
    <width>718</width>
    <height>535</height>
    <depth>3</depth>
  </size>
  <segmented>0</segmented>
  <object>
    <name>starbuckscoffee</name>
    <pose>Unspecified</pose>
    <truncated>0</truncated>
    <difficult>0</difficult>
    <bndbox>
      <xmin>466</xmin>
      <ymin>133</ymin>
      <xmax>709</xmax>
      <ymax>287</ymax>
    </bndbox>
  </object>
  <object>
    <name>starbucks-symbol</name>
    <pose>Unspecified</pose>
    <truncated>0</truncated>
    <difficult>0</difficult>
    <bndbox>
      <xmin>193</xmin>
      <ymin>270</ymin>
      <xmax>261</xmax>
      <ymax>360</ymax>
    </bndbox>
  </object>
  <object>
    <name>starbucks-symbol</name>
    <pose>Unspecified</pose>
    <truncated>0</truncated>
    <difficult>0</difficult>
    <bndbox>
      <xmin>420</xmin>
      <ymin>423</ymin>
      <xmax>464</xmax>
      <ymax>513</ymax>
    </bndbox>
  </object>
  <object>
    <name>six</name>
    <pose>Unspecified</pose>
    <truncated>0</truncated>
    <difficult>0</difficult>
    <bndbox>
      <xmin>633</xmin>
      <ymin>327</ymin>
      <xmax>671</xmax>
      <ymax>365</ymax>
    </bndbox>
  </object>
  <object>
    <name>tchibo</name>
    <pose>Unspecified</pose>
    <truncated>0</truncated>
    <difficult>0</difficult>
    <bndbox>
      <xmin>8</xmin>
      <ymin>312</ymin>
      <xmax>34</xmax>
      <ymax>351</ymax>
    </bndbox>
  </object>
</annotation>
```

Algorithms and Techniques

As mentioned before, we start by training an own CNN architecture to classify the logos as done in the dog breeds project of the Nanodegree. For this, 40% of the dataset forms

the test set, and the remaining data is split into 80% training set and 20% validation set. Using Keras preprocessing, each image is converted into a 4D tensor with shape $(1, 224, 224, 3)$. This CNN consists of a convolutionary layer, a max-pooling layer, another convolutionary layer, a max-pooling layer, a global-average pooling layer and a final fully connected layer. As optimizer we choose "RMSprop", as loss function "categorical cross entropy" and as metric "accuracy". Using Keras ModelCheckpoint, we save the model with the best validation loss.
As a next step we train a popular CNN like VGG19 whose weights are pre-trained on ImageNet. We proceed similarly as in the first setting.
Having tried CNNs for logo classification, we finally proceed to Faster R-CNNs for logo detection and classification. We first train a *Region Proposal Network (RPN)* that proposes regions with logos in images. The predicted region proposals are then reshaped using a *Region of Interest (RoI)* pooling layer. This layer is next used to classify the image within the proposed region and predict the offset values for the bounding boxes. For the latter task, we again train a CNN like VGG19 that is pre-trained on ImageNet. As explained above, here we use mean average precision as metric.

Benchmark

The recent *Logos in the Wild* dataset has not been studied much yet. When introduced in [THMB18], the focus is put on open set logo retrieval where only one sample image of a logo is available.
Instead we want to focus on a closed world assumption where we train and test on the Logos in the Wild dataset which has multiple images per brand. Therefore, we can only compare our results with the ones of other models that were trained and tested on the popular closed dataset *FlickrLogos-32* [RPLvZ11]. As cited in [THMB18], the mean average precision (map) in state-of-the-art results is 0.811 achieved by Faster-RCNN [SZG16] where the training set is expanded with synthetically generated training images, and 0.842 using Fast-M [BLF^{+}16] that is a multi-scale Fast R-CNN based-approach.

III. Methodology

(approx. 3-5 pages)

Data Preprocessing

Having gained access, we download the Logos in the Wild dataset [THMB18] that contains XML files with the bounding boxes, the URLs to the JPEG images, samples and a script to obtain a clean dataset. As stated above, we obtain the JPEG images of the Logos in the Wild dataset by downloading the superset QMUL-OpenLogo Dataset [SZG18].
After setting the variable "oldpath" to the absolute path of our `LogosInTheWild-v2/data` directory and the variable "new path" to the absolute path of our `openlogo/JPEGImages` directoy in our script move_JPEG_images.sh from the Logo Cap-

stone Project, we execute this script that moves the JPEG files from the `openlogo/JPEGImages` directory in the corresponding brand subdirectory in the `LogosInTheWild-v2/data` directory.

Next we execute the Python script create_clean_dataset.py from the `LogosInTheWild-v2/scripts` directory that adjusts the brand names in the XML files and removes XML files without corresponding JPEG image. This outputs that 9,428 images and 821 brands were processed, while 1,330 JPEG files were unavailable.

Finally, for Tensorflow's Object Detection API we need to convert our dataset with annotations in Pascal-VOC style to Tensorflow's TFRecord file format. For this, we adjusted their Python script

```
# From LogosInTheWild-v2 directory
python create_pascal_tf_record.py --data_dir=./data/
   voc_format --year=VOC2012  --label_map_path=./data/
   pascal_label_map.pbtxt --output_path=./data/

(py2) python object_detection/dataset_tools/
   create_pascal_tf_record.py --data_dir=/mnt/c/environment/
   Machine_Learning/Machine_Learning_Engineer/
   capstone_project/LogosInTheWild-v2/data/voc_format --year
   =VOC2012  --label_map_path=/mnt/c/environment/
   Machine_Learning/Machine_Learning_Engineer/
   capstone_project/LogosInTheWild-v2/data/pascal_label_map.
   pbtxt --output_path=/mnt/c/environment/Machine_Learning/
   Machine_Learning_Engineer/
capstone_project/LogosInTheWild-v2/data/
```

```
# From LogosInTheWild-v2 directory
python analyze_pascal_tf_record.py --data_dir=./data/
   voc_format --year=VOC2012  --label_map_path=./data/
   pascal_label_map.pbtxt --output_path=./data/

(py2) cd /mnt/c/ProgramData/anaconda3/envs/tensorflow/models
   /research
python object_detection/dataset_tools/
   analyze_pascal_tf_record.py --data_dir=/m
nt/c/environment/Machine_Learning/Machine_Learning_Engineer/
   capstone_project/LogosInTheWild-v2/data/voc_format --year
   =VOC2012  --label_map_path=/mnt/c/environment/Machine_L
earning/Machine_Learning_Engineer/capstone_project/
   LogosInTheWild-v2/data/pascal_label_map.pbtxt --
   output_path=/mnt/c/environment/Machine_Learning/
   Machine_Learning_Engineer
/
```

```
cd /mnt/c/ProgramData/anaconda3/envs/tensorflow/models/research
source ~/.bashrc
export PYTHONPATH=$PYTHONPATH:`pwd`:`pwd`/slim
PIPELINE_CONFIG_PATH=/mnt/c/environment/Machine_Learning/Machine_Learning_Engineer/capsto
MODEL_DIR=/mnt/c/environment/Machine_Learning/Machine_Learning_Engineer/capstone_project/
NUM_TRAIN_STEPS=20000
SAMPLE_1_OF_N_EVAL_EXAMPLES=1
TRAIN_DIR=/mnt/c/environment/Machine_Learning/Machine_Learning_Engineer/capstone_project/
EVAL_DIR=/mnt/c/environment/Machine_Learning/Machine_Learning_Engineer/capstone_project/L
python object_detection/model_main.py --pipeline_config_path=${PIPELINE_CONFIG_PATH} --mo


python object_detection/model_main.py --pipeline_config_path=C:\environment\Machine_Lear


python object_detection/legacy/train.py --logtostderr --train_dir=${TRAIN_DIR} --pipeline

 python object_detection/legacy/eval.py  --logtostderr   --pipeline_config_path=${PIPELII

YOUR_CLOUD_BUCKET=gs://logo-bucket-europe-west1//
PATH_TO_LOCAL_YAML_FILE=/mnt/c/environment/Machine_Learning/Machine_Learning_Engineer/cap
GS_MODEL_DIR=logo-bucket-europe-west1/LogosInTheWild-v2/models/model
GS_TRAIN_DIR=logo-bucket-europe-west1/LogosInTheWild-v2/models/model/train
GS_EVAL_DIR=logo-bucket-europe-west1/LogosInTheWild-v2/models/model/eval
GS_PIPELINE_CONFIG_PATH=logo-bucket-europe-west1/LogosInTheWild-v2/models/model/faster_rc
# From tensorflow/models/research/
gcloud ml-engine jobs submit training object_detection_`date +%m_%d_%Y_%H_%M_%S` --runtin


gcloud ml-engine jobs submit training object_detection_`date +%m_%d_%Y_%H_%M_%S` --runtin

gcloud ml-engine jobs submit training object_detection_`date +%m_%d_%Y_%H_%M_%S` --runtin


%train
  $ gcloud ml-engine jobs describe object_detection_04_29_2019_20_38_31

or continue streaming the logs with the command

  $ gcloud ml-engine jobs stream-logs object_detection_04_29_2019_20_38_31



 %eval
```

```
Maybe overwriting eval_num_epochs: 1
Maybe overwriting load_pretrained: True
Ignoring config override key: load_pretrained
Maybe overwriting train_steps: 10
Maybe overwriting sample_1_of_n_eval_examples: 1
Expected number of evaluation epochs is 1, but instead encountered ‘eval_on_train_input_c
```

```
gcloud auth application-default login
```

```
YOUR_CLOUD_BUCKET=gs://logo-bucket-europe-west1//
tensorboard --logdir=${YOUR_CLOUD_BUCKET}
```

```
tensorboard --logdir=${MODEL_DIR}
```

Implementation

In this section, the process for which metrics, algorithms, and techniques that you implemented for the given data will need to be clearly documented. It should be abundantly clear how the implementation was carried out, and discussion should be made regarding any complications that occurred during this process. Questions to ask yourself when writing this section:
Is it made clear how the algorithms and techniques were implemented with the given datasets or input data? Were there any complications with the original metrics or techniques that required changing prior to acquiring a solution? Was there any part of the coding process (e.g., writing complicated functions) that should be documented?

Refinement

In this section, you will need to discuss the process of improvement you made upon the algorithms and techniques you used in your implementation. For example, adjusting parameters for certain models to acquire improved solutions would fall under the refinement category. Your initial and final solutions should be reported, as well as any significant intermediate results as necessary. Questions to ask yourself when writing this section:
Has an initial solution been found and clearly reported? Is the process of improvement clearly documented, such as what techniques were used? Are intermediate and final solutions clearly reported as the process is improved?

IV. Results

(approx. 2-3 pages)

Model Evaluation and Validation

In this section, the final model and any supporting qualities should be evaluated in detail. It should be clear how the final model was derived and why this model was chosen. In addition, some type of analysis should be used to validate the robustness of this model and its solution, such as manipulating the input data or environment to see how the models solution is affected (this is called sensitivity analysis). Questions to ask yourself when writing this section:
Is the final model reasonable and aligning with solution expectations? Are the final parameters of the model appropriate? Has the final model been tested with various inputs to evaluate whether the model generalizes well to unseen data? Is the model robust enough for the problem? Do small perturbations (changes) in training data or the input space greatly affect the results? Can results found from the model be trusted?

Justification

In this section, your models final solution and its results should be compared to the benchmark you established earlier in the project using some type of statistical analysis. You should also justify whether these results and the solution are significant enough to have solved the problem posed in the project. Questions to ask yourself when writing this section:
Are the final results found stronger than the benchmark result reported earlier? Have you thoroughly analyzed and discussed the final solution? Is the final solution significant enough to have solved the problem?

V. Conclusion

(approx. 1-2 pages)

Free-Form Visualization

In this section, you will need to provide some form of visualization that emphasizes an important quality about the project. It is much more free-form, but should reasonably support a significant result or characteristic about the problem that you want to discuss. Questions to ask yourself when writing this section:
Have you visualized a relevant or important quality about the problem, dataset, input data, or results? Is the visualization thoroughly analyzed and discussed? If a plot is provided, are the axes, title, and datum clearly defined?

Reflection

In this section, you will summarize the entire end-to-end problem solution and discuss one or two particular aspects of the project you found interesting or difficult. You are expected to reflect on the project as a whole to show that you have a firm understanding of the entire process employed in your work. Questions to ask yourself when writing this section:
Have you thoroughly summarized the entire process you used for this project? Were there any interesting aspects of the project? Were there any difficult aspects of the project? Does the final model and solution fit your expectations for the problem, and should it be used in a general setting to solve these types of problems?

Improvement

In this section, you will need to provide discussion as to how one aspect of the implementation you designed could be improved. As an example, consider ways your implementation can be made more general, and what would need to be modified. You do not need to make this improvement, but the potential solutions resulting from these changes are considered and compared/contrasted to your current solution. Questions to ask yourself when writing this section:
Are there further improvements that could be made on the algorithms or techniques you used in this project? Were there algorithms or techniques you researched that you did not know how to implement, but would consider using if you knew how? If you used your final solution as the new benchmark, do you think an even better solution exists? WARNING:tensorflow:Ignoring ground truth with image id 53383224 since it was previously added WARNING:tensorflow:Ignoring detection with image id 53383224 since it was previously added creating index... index created! creating index... index created! Running per image evaluation... Evaluate annotation type *bbox* DONE (t=52.51s). Accumulating evaluation results... Segmentation fault (core dumped)

References

[BBMS17] Simone Bianco, Marco Buzzelli, Davide Mazzini, and Raimondo Schettini. Deep learning for logo recognition. *Neurocomputing*, 245:23 – 30, 2017.

[BLF+16] Yu Bao, Haojie Li, Xin Fan, Risheng Liu, and Qi Jia. Region-based cnn for logo detection. In *Proceedings of the International Conference on Internet Multimedia Computing and Service*, ICIMCS'16, pages 319–322, New York, NY, USA, 2016. ACM.

[EVGW+10] Mark Everingham, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. The pascal visual object classes (voc) challenge. *International journal of computer vision*, 88(2):303–338, 2010.

[ISGK15] Forrest N. Iandola, Anting Shen, Peter Gao, and Kurt Keutzer. Deeplogo:

Hitting logo recognition with the deep neural network hammer. *CoRR*, abs/1510.02131, 2015.

[ker] Keras Documentation. `https://keras.io/applications/`. Online; accessed 2019-03-17.

[KPT+11] Y. Kalantidis, LG. Pueyo, M. Trevisiol, R. van Zwol, and Y. Avrithis. Scalable triangulation-based logo recognition. In *in Proceedings of ACM International Conference on Multimedia Retrieval (ICMR 2011)*, Trento, Italy, April 2011.

[RHGS15] Shaoqing Ren, Kaiming He, Ross B. Girshick, and Jian Sun. Faster R-CNN: towards real-time object detection with region proposal networks. *CoRR*, abs/1506.01497, 2015.

[RPLvZ11] Stefan Romberg, Lluis Garcia Pueyo, Rainer Lienhart, and Roelof van Zwol. Scalable logo recognition in real-world images. In *Proceedings of the 1st ACM International Conference on Multimedia Retrieval*, ICMR '11, pages 25:1–25:8, New York, NY, USA, 2011. ACM.

[SZG16] Hang Su, Xiatian Zhu, and Shaogang Gong. Deep learning logo detection with data expansion by synthesising context. *CoRR*, abs/1612.09322, 2016.

[SZG18] Hang Su, Xiatian Zhu, and Shaogang Gong. Open logo detection challenge. *CoRR*, abs/1807.01964, 2018.

[THMB18] Andras Tüzkö, Christian Herrmann, Daniel Manger, and Jürgen Beyerer. Open Set Logo Detection and Retrieval. In *Proceedings of the 13th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications: VISAPP*, 2018.