Documentation: Parking detection

This document describes the methodology used to detect parking lots in satellite images. The software contained in the repository deals with three main steps:

1. Image pre-processing and segmentation
2. Dataset generation: training and validation cookie set
3. Deep Neural Network (googlenet and alexnet) trained on the cookies

# Image pre-processing and segmentation

The input data consisted of (i) one geotiff image and (ii) one shapefile containing 36 polygons indicating the parking lots in the image. A python application was built to:

* load the two files
* generates a range of (line, pixel) containing each polygon in the shapefile
* segment the image in cookies of size 256X256
* identify the cookies overlapping the polygon area to label them as parking cookies (label 1)
* save 2\*N cookies (N positive cookies with parking and N negative cookies without parking)

*Gdal* and *shapefile* python libraries were installed to load and process the two files.

The image is segmented into 256x256 cookies with overlapping window of 64 pixels. This allowed to increase the number of generated parking cookies. The segmentation has an offset of 200 pixels: the first 200 pixels of the geotiff were black and did not contain any information.

# Cookie labelling

A cookie is labelled as parking cookie if it lies (partially or fully) within the boundaries of a polygon. The [ray casting algorithm](https://en.wikipedia.org/wiki/Point_in_polygon) is used to identify whether the pixels of the cookie are inside one of the polygons. A cookie is labelled as parking if at least 100 pixels are within a parking polygon.

# Dataset generation

The resulting dataset contains 2608 cookies. The data is divided into training and validation sets with the proportion of 80% and 20% respectively, taking negative samples uniformly at random:

* training set: file sample\_train.csv with 1956 samples
* test set: file sample\_val.csv with 652 samples

# Deep Learning

GoogLeNet/Inception and AlexNet were used to detect parking lots. These networks were successfully used in the 2012 and 2014 [Large Scale Visual Recognition Challenge](http://www.image-net.org/challenges/LSVRC/) for classifying images among 200 classes. They have recently used for classifying, detecting and counting cars from satellite images ([Nathan et al.](http://link.springer.com/chapter/10.1007/978-3-319-46487-9_48)).

I tested the GoogLeNet and AlexNet implemented in [chainer](https://github.com/pfnet/chainer/tree/master/examples/imagenet). They require input as 256x256 RGB images. A mean cookie is computed from the training set and saved as mean\_256.npy numpy array. This dataset is used for the training of the two deep neural networks.

As an initial test, I trained the networks with a smaller sample size:

* a training set with 148 samples
* a validation set with 37 samples

The learning rate was set at lr=0.01 (similar results were observed for different values). Results (fig. 1 and fig. 2). Due to the small sample size, the networks were trained with a batch size of 16 and 32. While the accuracy does not change (probably due to the limited training data), the loss decreases more rapidly when the training batch size is set to 32.

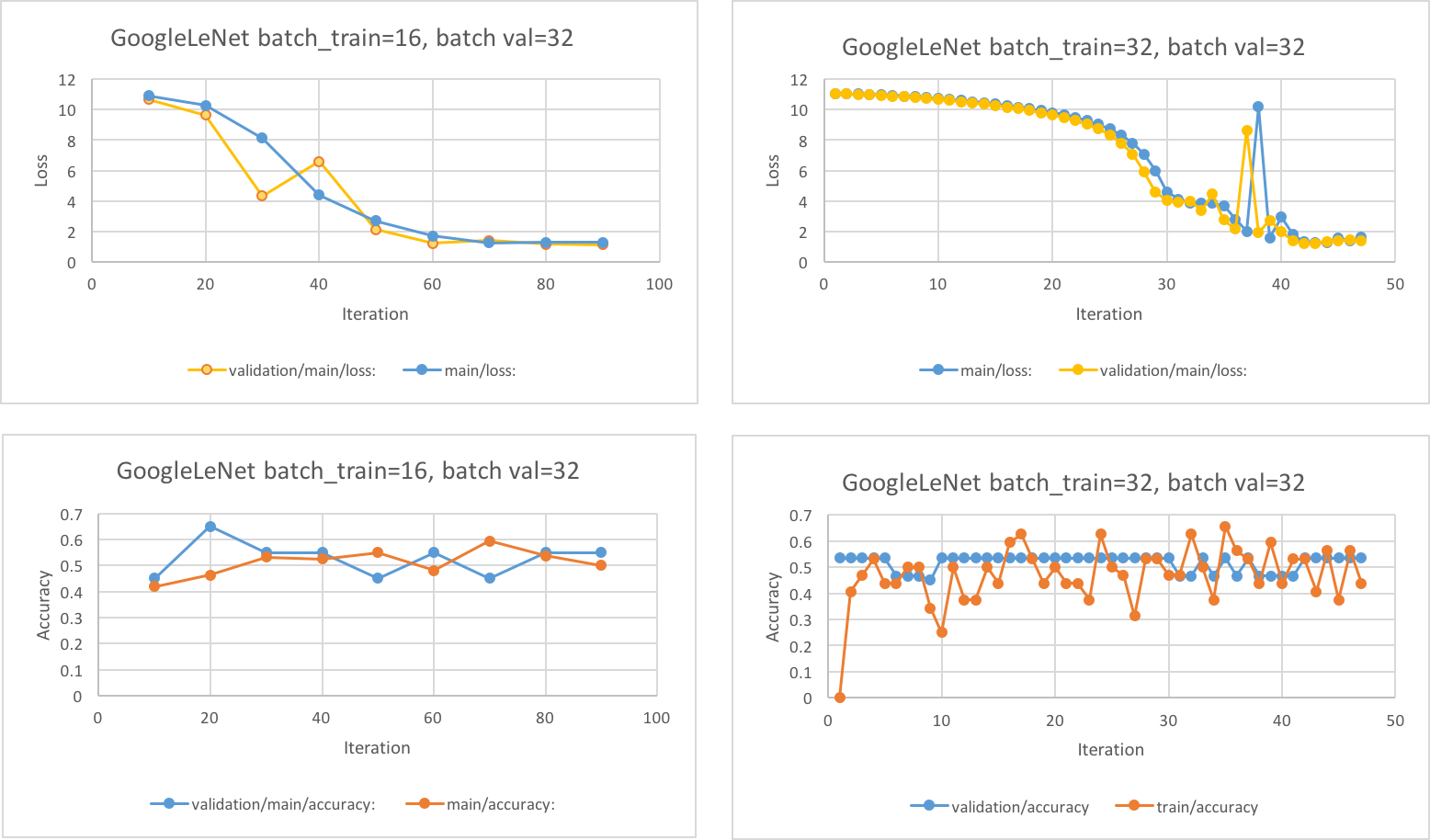


Figure 1: GoogLeNet loss and accuracy over the iterations

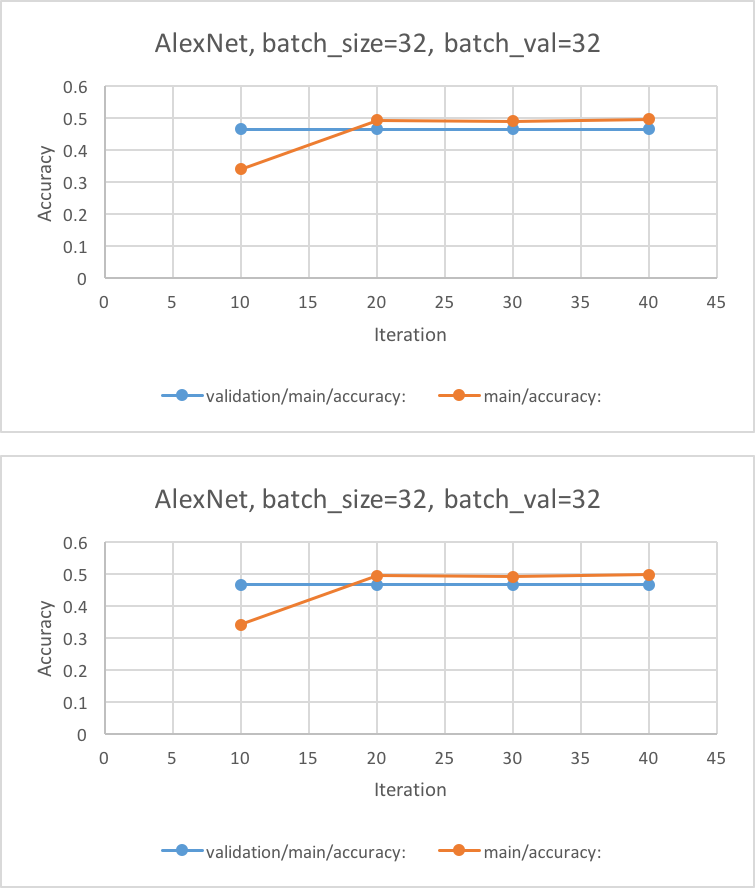


Figure 2: AlexNet loss and accuracy over the iterations

Given the previous results, a longer training was performed on the entire dataset using GoogLeNet with batch = 32. The results described in the table below show the same decreasing trend in the loss but not improvements in accuracy (again, probably due to the small training set). Interestingly, the accuracy is comparable between the training and validation set, thus showing no signs of overfitting.

epoch iteration main/loss validation/main/loss main/accuracy validation/main/accuracy lr

0 1 11.0513 11.0417 0 0.506944 0.01

0 2 11.0417 11.0245 0.59375 0.56498 0.01

0 3 11.0238 11.0005 0.59375 0.506448 0.01

0 4 11.0002 10.9698 0.46875 0.506448 0.01

0 5 10.9692 10.9329 0.59375 0.506448 0.01

0 6 10.9345 10.8899 0.34375 0.506448 0.01

0 7 10.8932 10.8415 0.3125 0.506448 0.01

0 8 10.8398 10.7878 0.625 0.506448 0.01

0 9 10.7876 10.7291 0.375 0.506448 0.01

0 10 10.7309 10.6657 0.34375 0.50496 0.01

0 11 10.6688 10.5974 0.5 0.515873 0.01

0 12 10.5955 10.5239 0.4375 0.493552 0.01

0 13 10.5263 10.4452 0.53125 0.493552 0.01

0 14 10.4482 10.3607 0.46875 0.493552 0.01

0 15 10.3605 10.2698 0.65625 0.493552 0.01

0 16 10.2691 10.1713 0.5 0.493552 0.01

0 17 10.1766 10.0641 0.28125 0.493552 0.01

0 18 10.061 9.94668 0.5625 0.493552 0.01

0 19 9.94284 9.81668 0.53125 0.493552 0.01

0 20 9.81717 9.67147 0.5 0.493552 0.01

0 21 9.64948 9.50639 0.59375 0.493552 0.01

0 22 9.48797 9.31574 0.53125 0.493552 0.01

0 23 9.32545 9.09199 0.4375 0.493552 0.01

0 24 9.07099 8.82183 0.625 0.493552 0.01

0 25 8.85125 8.48759 0.5 0.493552 0.01

0 26 8.50318 8.05673 0.5 0.493552 0.01

0 27 8.08592 7.47299 0.53125 0.493552 0.01

0 28 7.48766 6.63184 0.4375 0.493552 0.01

0 29 6.61547 5.39477 0.40625 0.493552 0.01

0 30 5.41162 4.1997 0.625 0.493552 0.01

0 31 4.26687 3.84789 0.5 0.493552 0.01

0 32 3.95323 3.58375 0.40625 0.506448 0.01

0 33 3.65076 3.09037 0.46875 0.506448 0.01

0 34 3.13055 2.53139 0.5 0.493552 0.01

0 35 2.60829 2.2427 0.4375 0.493552 0.01

0 36 2.54578 7.41448 0.4375 0.506448 0.01

0 37 8.40873 13.6965 0.4375 0.493552 0.01

0 38 11.4344 6.94649 0.5625 0.493552 0.01

0 39 5.9868 3.26151 0.59375 0.493552 0.01

0 40 2.60101 2.03593 0.625 0.493552 0.01

0 41 1.91272 1.67924 0.53125 0.493552 0.01

0 42 1.64403 1.68831 0.53125 0.493552 0.01

0 43 1.93266 1.73242 0.34375 0.506448 0.01

0 44 1.75076 1.74312 0.53125 0.506448 0.01

0 45 1.7084 1.74082 0.59375 0.506448 0.01

0 46 1.6864 1.76049 0.59375 0.506448 0.01

0 47 1.60792 1.80037 0.59375 0.506448 0.01

0 48 1.95259 1.79869 0.46875 0.506448 0.01

0 49 2.01131 1.7304 0.4375 0.506448 0.01

0 50 1.43879 1.66922 0.59375 0.506448 0.01

0 51 1.52984 1.60086 0.5625 0.506448 0.01

0 52 1.80817 1.48279 0.4375 0.506448 0.01

0 53 1.58738 1.33953 0.40625 0.506448 0.01

0 54 1.22423 1.21325 0.65625 0.506448 0.01

0 55 1.28677 1.17236 0.40625 0.506448 0.01

0 56 1.1865 1.19812 0.5 0.493552 0.01

0 57 1.26212 1.23942 0.46875 0.493552 0.01

0 58 1.30404 1.26266 0.46875 0.493552 0.01

0 59 1.14803 1.29877 0.625 0.493552 0.01

0 60 1.36049 1.32991 0.46875 0.493552 0.01

0 61 1.21098 1.32359 0.59375 0.493552 0.01

1 62 1.20911 1.26256 0.5625 0.493552 0.01

1 63 1.2868 1.17621 0.46875 0.493552 0.01

1 64 1.19511 1.14828 0.5625 0.493552 0.01

1 65 1.2045 1.14084 0.375 0.506448 0.01

1 66 1.17293 1.13206 0.4375 0.506448 0.01

1 67 1.12247 1.14529 0.53125 0.506448 0.01

1 68 1.23234 1.17244 0.40625 0.506448 0.01

1 69 1.21056 1.16512 0.53125 0.506448 0.01

# Next Steps

Given the poor accuracy, a natural next step is to significantly increase the training samples to a play with the cookie size to potentially provide more focussed training images. Other network architectures such as ResNet or ResCeption might also be explored as potentially competitive candidates.