Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Finding Advanced Features for Housing Price Prediction

Bachelor Thesis

Ali Uzpak

uzpaka@student.ethz.ch

Distributed Computing Group
Computer Engineering and Networks Laboratory
ETH Zürich

Supervisors:
Gino Brunner, Yuyi Wang
Prof. Dr. Roger Wattenhofer

August 19, 2017

Acknowledgements

I thank my supervisors Gino Brunner and Yuyi Wang for their help and advice throughout this work. I also thank the Computer Engineering and Networks Laboratory (TIK) for providing the necessary GPU servers.

Abstract

In this thesis, we try to find new features that could be used for predicting housing prices. In particular, we focus on visual features such as satellite images. With the help of free existing APIs, we automatically generate our own dataset of high resolution satellite images and labels for each municipality in Switzerland and Germany. We then use various neural network architectures to predict data such as the number of people or houses present in those images. Finally, we train and use neural networks that are able to do road segmentation.

Contents

Chapter 1

Introduction

1.1 Motivation

Nowadays, the real-estate market is under pressure and is a concern in most modern countries and cities in the world. In Switzerland for example, housing prices seem to consistently increase in every region of the country. As a consequence, being able to determine or predict real-estate prices has become increasingly important.

The classical and natural approach to predict housing prices would be to collect "standard" housing features, such as location, age of the building, size of the living space, number of rooms, etc. and then use a regression model such as Support Vector Machines.

However, as noted in [1, 2], it is likely that a house's price is not only determined by those standard features. One also has to consider, for example, the house's neighbourhood and its accessibility to useful infrastructures or facilities (roads, schools, train stations, shops).

The goal of this thesis is to provide a basis to actually find those new, non-standard features which could directly or indirectly help predict real-estate prices. Namely, one approach would be to examine the satellite image of a house in order to see whether or not the house is well located and has access to many facilities.

Due to the problem complexity and lack of data, we will focus on simpler tasks that are not directly related to housing prices but could be a basis for it. Our first task consists of predicting the population and the number of houses in a city given its satellite image (Chapter 3). Our second task consists of road detection and segmentation (Chapter 4).

Furthermore, it can sometimes be difficult to get high resolution satellite pictures (and labels) of a particular city or place. In this thesis, we also present a way to automatically get high resolution satellite images and labels using the Google Static Maps API and the OpenStreetMap Nominatim API (Chapter 2).

1.2 Related Work

Housing price prediction is a challenging task that has already been studied using Machine Learning. In [3], 200 houses in New Zealand were selected and used in a hedonic price model and an artificial neural network (ANN) model. It was empirically shown that the ANN model was more accurate. In [1], spatial features of houses (i.e, their locations and accesses to infrastructures or facilities) were used to improve the price prediction on a dataset of 66'000 transaction records. Likewise, in [2], the use of visual features contained in pictures of houses improved the price prediction on a dataset of 535 samples.

There is also a lot of research in the area of satellite images learning. Namely, in [4], a deep convolutional network followed by a post-processing SVM-based filter is used to predict road/non-road patches.

In [5], new loss functions are introduced to improve the accuracy of convolutional networks for road and building segmentation. One of the datasets used (the Massachusetts Roads and Buildings Dataset [6]) has also been made public, which was helpful for developing our own model in Chapter 4.

Chapter 2

Datasets Generation

We are interested in satellite images of each municipality in Switzerland and Germany.

For Switzerland, the official Federal Statistical Office (Bundesamt für Statistik, or BfS) [7] provides a list of all Swiss municipalities and their number of inhabitants for the year 2016. However, we could not find data about the number of buildings per municipality.

For Germany, we used data provided by the "Statistische Ämter des Bundes und der Länder" [8], which contains each German municipality and their number of inhabitants and buildings. The data about the number of inhabitants is recent (2016), but the data about the number of buildings is from 2009, which could be a bit outdated.

In total, the Swiss and German datasets contain 6810 and 11167 entries, respectively.

2.1 Direct and Naïve Approach

Given the lists of municipalities, one can use a Static Map API to get the satellite images of the cities. There are many (free and non-free) Static Map APIs available, and we decided to use the one from Google [9] because it allows 25'000 queries a day and gives high-resolution RGB images (1280x1280). Figure 2.1 shows a satellite image of the city of Zürich by directly using the Google API with the following URL:

`http://maps.googleapis.com/maps/api/staticmap?&size=640x640&zoom=13&center=zurich&maptype=satellite`

We can see from the URL that, in Figure 2.1, a zoom of 13 was used. This parameter is important for getting good quality images for subsequent training and predictions.

In particular, the fact that we are limited to 1280x1280 images leads to the following trade-off: By increasing the zoom, we get "better" satellite images in

the sense that we would be able to see more details in the pictures. However, the resulting satellite picture would not contain the entire city, but only a part of it. This may be problematic because our population and houses data are given for entire cities only.

Moreover, even if the zoom is small enough so that the image contains the entire city, we still have a problem. Namely, the image would now contain more than just the city that we are interested in, because parts of the surrounding municipalities will appear as well.

For all these reasons, this direct and naïve approach results in data that does not perfectly correspond to the ground truth. Therefore, we decided to generate an additional dataset by using a new approach developed in Section 2.2.

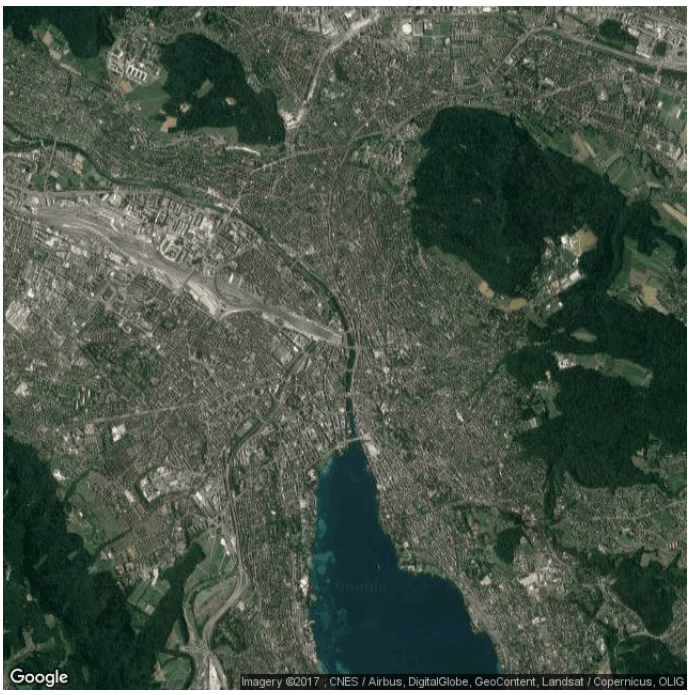
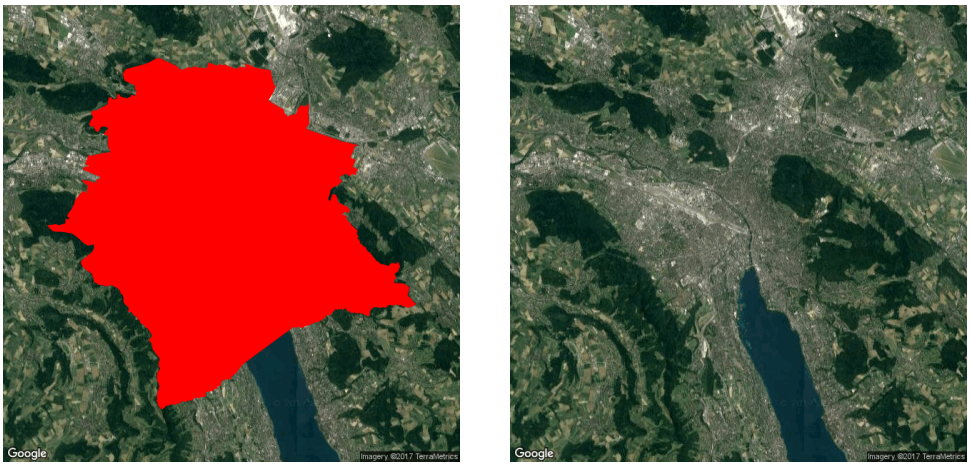


Figure 2.1: Satellite map of Zürich, with a zoom of 13

2.2 Indirect Approach with OpenStreetMap

To get ground truth data, we need to make sure that our satellite images only show what is inside the legal boundaries of the municipalities. Unfortunately, the Google Static Map API does not give boundaries for the queried municipalities and thus cannot be directly used to extract ground truth data.

However, there are other sources that have data about cities' boundaries, like the OpenStreetMap (OSM) Nominatim API [10]. For each municipality



(a) Zürich's boundaries (red mask) (b) Zürich's boundaries (transparent mask)

Figure 2.2: Maps of Zürich using the Google API, unfixed zoom

queried, this API gives its legal geographical boundaries as a polygon, i.e, a list of (latitude, longitude) points such that the last and first points equal.

Given this data-structure, we can now use the Google API and query the polygon (i.e the city's boundaries) directly. For example, Figure 2.2a can be obtained with the following URL:

`https://maps.googleapis.com/maps/api/staticmap?maptype=satellite&size=640x640&scale=1&path=weight:0|color:0x000000ff|fillcolor:0xff0000ff|enc:ENCODED_POLYGON`

It should be noted that the URLs used in the Google API have a size limit of 8192 characters. Therefore, the polygons given by the OSM API have to be either encoded or downsampled (or both, if necessary). We used the encoding algorithm provided by the 'Polyline' Python package [11, 12] and only downsampled the polygons that were too big even after encoding.

Given the 2 images of Figures 2.2a and 2.2b, one can easily remove the region (outside the red mask) that does not belong to the municipality we queried. This would result in a new image shown in Figure 2.3.

Unfortunately, this new approach has its downsides as well. The first issue is that the generated images will not have a consistent scale or zoom between them. In fact, as we only query each city's polygon, the zoom will be adapted consequently (this is automatically done by the Google API). For example, to fit the entire city of Zürich in a 1280x1280 image, the zoom will have to be relatively low. On the other hand, a small village can fit in a 1280x1280 image with a relatively high zoom. Moreover, the fact that the zoom is adapted to the municipalities' sizes leads to an additional issue that concerns large cities such as Zürich: As the zoom has to be low for such cities, one can barely see

Figure 2.3: Satellite map of Zürich without the surrounding municipalities

and distinguish the different buildings and roads in the queried satellite image (see Figure 2.3 for example). This could potentially make the tasks of detecting roads and counting population or buildings more difficult.

One approach to overcome these issues would be to have a fixed and high zoom while querying the cities' polygons. That way, all our images in the dataset will have a consistent scale, and the buildings or roads in satellite images of large cities will be easy to see and distinguish. However, this can again result in images that do not perfectly represent the cities, as one can see in Figure 2.4b. In fact, as we are limited to 1280x1280 pixels, a high zoom for a large city will result in a satellite image that might not contain the entire city, but only a part of it, like its centre for example. We could still just filter out the municipalities that do not fit in 1280x1280 pixels, but the resulting dataset would be reduced and would only contain small cities.

Given all these trade-offs, we decided to use both fixed and unfixed zooms, without filtering out cities that do not fit in the pictures.

2.3 Labels Generation for Road Detection

For our last task in Chapter 4, we will need "label-images" for each satellite image in our dataset. The label-image maps each original pixel to its class, i.e,

(a) (b)

Figure 2.4: Label and satellite maps of Zürich, zoom=14

in our case, a value saying whether that pixel is a road or not.

This pixel-wise labelling can be done again with the Google Static Map API. For example, we can generate the label image shown in Figure 2.4a by using the following URL:

```
http://maps.googleapis.com/maps/api/staticmap?&size=640x640&zoom=
14&center=zurich&scale=1&maptype=roadmap&style=feature:road|element:
geometry.fill|color:0xffff00&style=feature:poi.park|element:geometry.
fill|color:0x00ff00&style=feature:landscape.natural|element:geometry.
fill|color:0x00ff00&&style=feature:all|element:labels|visibility:
off&style=feature:transit|element:geometry|visibility:off&style=feature:
water|element:geometry.fill|color:0x0000ff
```

An interesting point is that the Google API can be used for more than just road segmentation. In fact, one could get labels for buildings, train stations, water, parks or green areas. However, we have noticed that these labels were often missing for small municipalities, this is why we focused on road segmentation only.

Chapter 3

Prediction of Population and Number of Houses

The prediction of population and number of houses is divided in 2 subtasks. First, in Section 3.1, we perform a binary classification by predicting whether a city is small (class 0) or big (class 1). Then, in Section 3.2, the task consists of regression, i.e, we directly predict the target value (number of houses) for each city.

For each task, we used Python 2.7 [12] with the deep learning library Keras 1.2.2 [13], running on top of Theano 0.9 [14]. The GPU used was the Nvidia GeForce GTX 1080 [15].

It is crucial to note that, in this chapter, all the experiments used satellite pictures with a fixed zoom of 15. If we let the zoom unfixed, our network was not able to perform better than random guessing in the classification task for example. In our opinion, this is due to the fact that, for many large cities, the corresponding satellite pictures were not of good quality because of the limited resolution and low zoom, as explained in Section 2.2.

3.1 Binary Classification of Cities

The classification is done on the number of inhabitants only. We first choose a threshold value (or classification point), and each city with a population below the threshold is considered as "small", otherwise it is considered as "big". The choice of the classification point is important, because it directly affects the dataset's distribution (i.e, the number of samples in each class). As we wanted a balanced dataset, in order to prevent our models to have a high bias towards the majority class, we opted for a classification point that corresponds to the median value of the population in the Swiss dataset. This median value was in our case 1447. In the German dataset, 46% of the municipalities have a population below 1447, which we considered as well balanced.

To help finding a good classification point, one could look at the distribution

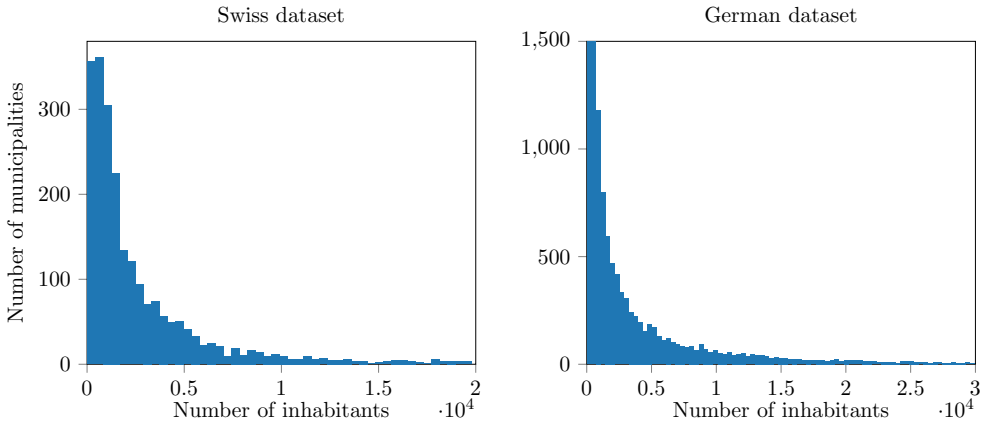


Figure 3.1: Histogram of the population

of the population and choose a point that clearly distinguishes small cities from big ones. However, as shown in Figure 3.1, it is not clear which classification point would be the best, as the distributions of our data appear to be long-tailed.

Before training the neural network, we resized each image in our dataset to 512x512 pixels and conserved the 3 (RGB) color channels. For the preprocessing, we only performed mean subtraction and division by variance. The architecture of our neural network is shown in Figure 3.2, and the model is compiled with the categorical cross entropy loss and the Adadelta [16] optimizer with default parameters.

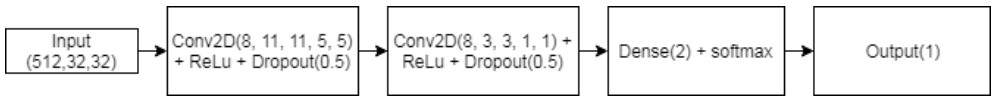


Figure 3.2: Architecture of our CNN in the binary classification task[1]

Table 3.1 shows the results of the following experiments: We first train our network on 80% of the Swiss dataset and test it on the remaining 20% during 100 epochs, and we do the same for the German dataset. Then, we train our network on the entire Swiss dataset and test it on the entire German dataset (and vice-versa). Unsurprisingly, our model performs better when the training data is of the same country than the test data. However, in all cases our model still performs better than random classification (≈ 0.5) and thus we can conclude that our network is able to generalize well, even if the training data is of a different country than the test data.

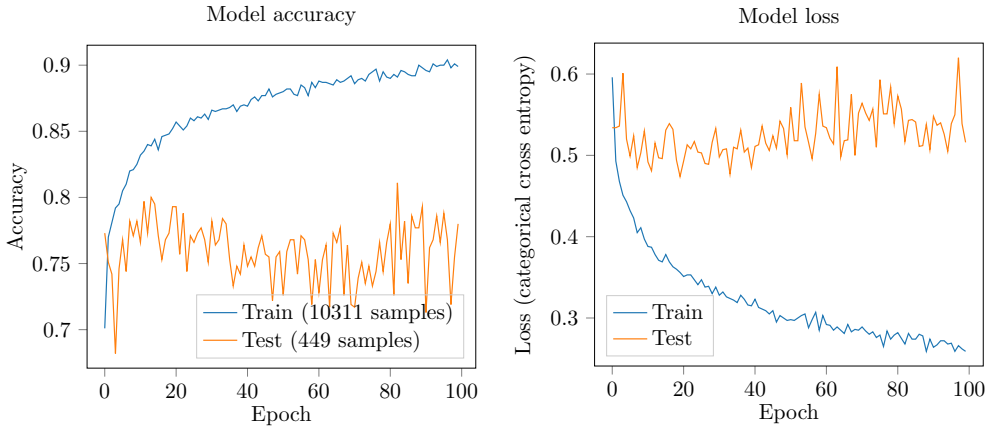
Finally, in our last experiment, we train our network on the entire German

[1] 'Conv2D(n,k1,k2,s1,s2)' denotes a 2D Convolution layer of n feature maps with a kernel size of (k1,k2) and a stride of (s1,s2). The layers' names are faithful to the documentation of Keras [13].

Tested on		Swiss	German
Trained on	Swiss	0.8351 at epoch 39	0.7209 at epoch 5
	German	0.7606 at epoch 48	0.8063 at epoch 48
	German and Swiss	0.8106 at epoch 82	N/A

Table 3.1: Best accuracy on the given test set when trained during 100 epochs on the given training set.

dataset and 80% of the Swiss dataset, and test it on the remaining 20% of the Swiss dataset during 100 epochs. As shown in Table 3.1, the best accuracy on the test set is 0.8106 at epoch 82. We note that having German data in our training set worsened the accuracy on the Swiss test set, as the accuracy on the Swiss test set is higher when the model is trained on the Swiss training set only (0.8351 versus 0.8106, see Table 3.1). For the model trained on both Swiss and German data, we show the accuracy and loss plots over the training epochs in Figure 3.3, with the corresponding confusion matrix. Again, that model is able to generalize well, as it performs better than random classification. Moreover, in Figure 3.4, we show the output of one feature map of the 2nd Convolution layer, given an unseen test sample: It is interesting to note that our network activates well for buildings and roads in our image, which are indeed relevant for determining the city's class (big or small).



Predicted		Small	Big	Total
Actual	Small	159	40	199
	Big	45	205	250
	Total	204	245	449

Figure 3.3: Model accuracy, loss and confusion matrix (on test set). $Training\ set = German\ data + 0.8 * Swiss\ data$. $Test\ set = 0.2 * Swiss\ data$

(a) Input data (b) Output of the 2nd intermediate layer

Figure 3.4: Satellite image and corresponding feature map for a Swiss test sample. Model trained on the entire German dataset and 80% of the Swiss dataset.

3.2 Regression for the Number of Houses

The regression is done on the number of houses or buildings only. Thus, we will only consider the German dataset, as we did not find data about the number of buildings for Swiss municipalities. Our training set contains 80% of our data (4949 samples) and the test set contains the remaining 20% (1238 samples). As in Section 3.1, we resize each image to 512x512 pixels and conserve the RGB channels. The preprocessing simply consists of mean subtraction and division by variance.

We present two approaches: the first uses a CNN [17] where the weights are initialized at random. The second uses a pre-trained model based on VGG-16 [18]. Both approaches only consider municipalities with a number of houses smaller or equal to 1000. This filtering is done to avoid the presence of large cities in our dataset, because, with a fixed zoom, the corresponding satellite images may only contain the cities' centers. This would have resulted in data that deviates too much from the ground truth.

In the first approach, our model is based on the SegNet [19] architecture. SegNet is primarily used for semantic segmentation (i.e, the task of predicting each pixel's class in an image) and consists of an encoding and decoding part. In our case, we are only predicting 1 continuous value and thus will only consider the encoding part of SegNet. The architecture of our model can be seen in more details in Figure 3.5. We used the Mean Squared Error (MSE) loss function and the Adam [20] optimizer with default parameters.

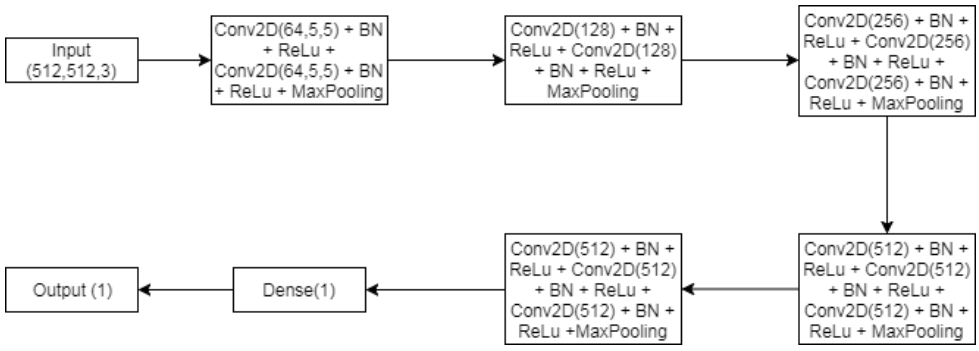
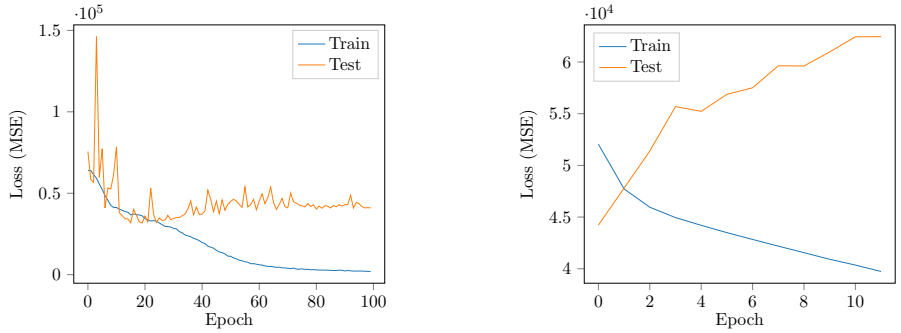


Figure 3.5: Architecture of our convolutional neural network in the first approach[2]

We trained the model during 100 epochs. Figure 3.6a shows the loss (MSE) plot of our model. The best loss on the test set was 31752.825 at epoch 20, which corresponds to a root MSE (RMSE) of 178.19. As the mean target value in the test set is 355.56, we conclude that our error is still too high and thus our model has not been able to perform well. The network was eventually able to fit the training set, and overfitting starts at the 20th epoch.



(a) First approach (b) Second approach (pre-trained model)

Figure 3.6: Model loss

In the second approach, we use transfer learning by taking the weights of the pre-trained VGG-16 model. The motivation behind this is that the VGG-16 model could be used as a feature extractor. The extracted features could then be fed to a fully connected network. The architecture of our model is similar to the one we used in the first approach, except that, this time, no batch normalizations are used and all the convolution kernels are of size (3,3). Moreover, after the

[2] 'Conv2D(n,k1,k2)' denotes a 2D Convolution layer of n feature maps with a kernel size of (k1,k2) and a stride of (1,1). When not specified, the kernel size is (3,3). 'BN' means Batch Normalization and 'MaxPooling' denotes a 2D Pooling layer with a downsampling factor of (2,2).

last 2D Convolution layer, we added a Dense layer with 300 units initialized at random, followed by a ReLu activation function. All the other layers of our network are initialized with the VGG-16 weights and frozen.

Figure 3.6b shows the loss plot of our second model. The results are even worse than in the first approach, as the best loss on the test set is 44221.293 at the first epoch. In fact, the network overfits already from the beginning and is not able to generalize at all; we stopped the training at epoch 10 because it was clear that the network would not be able to achieve good performance. Moreover, we also tried to use unfrozen VGG-16 weights (i.e, weights that can still be updated during training), but that did not help decreasing the loss and therefore we do not show those results here.

First approach	31752.825 at epoch 20
Second approach (pre-trained model)	44221.293 at epoch 1

Table 3.2: Best Mean Squared Error loss on the test set for both approaches.

For the regression task, we present a summary of our results in Table 3.2. We have also tried many different settings, architectures and parameters, but we always got poor performances. In our opinion, this could be explained by the fact that our dataset contains either bad quality images, or incomplete ones, due to the limitation of 1280x1280 pixels by the Google API. Furthermore, we think that the target values (number of houses) should be manually measured to make sure of the precision and correctness of our data.

Chapter 4

Semantic Segmentation for Road Detection

Road detection or extraction can be seen as a semantic segmentation task, in which one has to predict the class (here: road or not) of each input pixel. This is nothing but a multi-dimensional classification problem in the sense that, for each input sample, our goal is to predict $n * m$ classes instead of just 1 (n and m are the width and height of the image, respectively).

Our model's architecture is again based on SegNet: For the encoding part, we initialize the weights with the one from VGG-16, but let them "unfrozen" (i.e, the weights can still be updated during training). For the decoding part, we initialize the weights at random and use fewer layers than in the original SegNet model. The encoding architecture is described in Chapter 3.2 and Figure 3.5, and our decoding architecture is shown in Figure 4.1.

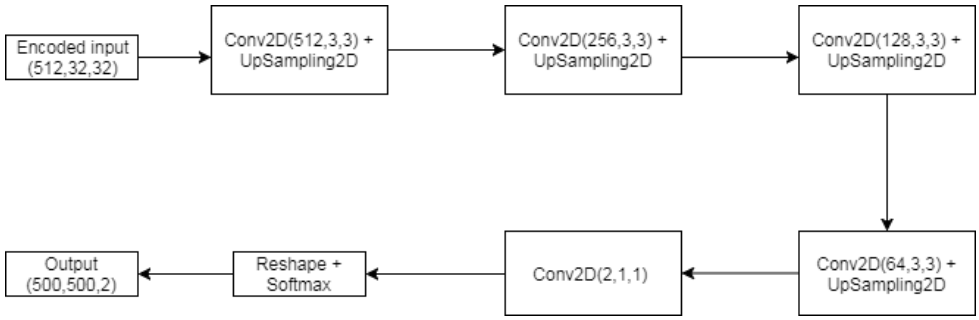


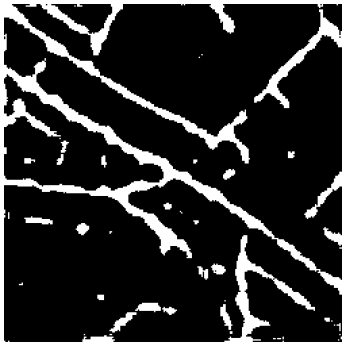
Figure 4.1: Architecture of our decoder in the road segmentation task[1]

For classification tasks, one would be tempted to use "intuitive" metrics such as the accuracy of prediction and the categorical cross entropy loss. In our case, one has to be really careful because our classes are highly unbalanced. In fact, a

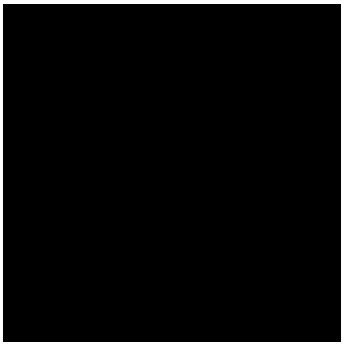
[1] 'UpSampling2D' upsamples the feature maps with a factor of (2,2). For clarity, we omitted the padding layers necessary to get the right number of pixels in the output (500*500). See Figure 3.5 for the encoder.

(a) Input image from the test set

(b) Target labels (ground truth)



(c) Model's prediction. Accuracy ≈ 0.90



(d) Black prediction. Accuracy ≈ 0.93

Figure 4.2: Satellite image with corresponding labels and predictions (Swiss dataset)

satellite image contains usually many more non-road pixels than road ones: In our dataset for example, more than 90% of the pixels are non-road. This can lead to the following issue: Figure 4.2a shows an input test sample from the Swiss dataset, and Figure 4.2b is its corresponding labels-image. The prediction of one of our models, shown in Figure 4.2c, is clearly better than the the black prediction of Figure 4.2d. However, the accuracy of the black prediction is 0.932564, while the accuracy of our model's prediction is 'only' 0.907964, which seems counter-intuitive.

To make sure that our network does not only predict black pixels, we assign weights (or costs) for each class in the loss function, such that a misclassification of a road-pixel is more penalized (i.e, it has a greater cost) than a misclassification of a non-road pixel. This has greatly improved the quality of our models' predictions.

For this task, we consider two datasets: The first one is the "Massachusetts Roads Dataset" published in [5, 6], which contains 1171 RGB satellite images of size 1500x1500 pixels. For each image, we generate 9 patches of 500x500 pixels and apply a histogram equalization on each colour channel to enhance the

contrast. Moreover, we augment the training data by rotating each patch by 90 degrees four times. The second dataset contains the Swiss satellite images that we generated in Section 2.2. For each 1280x1280 satellite image in the Swiss dataset, we only consider the inner four 500*500 pixels patches. That way, our model can easily be used for both datasets without any modification of its parameters (such as the input shape of (500,500,3)). Histogram equalization was also applied on each colour channel of the Swiss image patches.

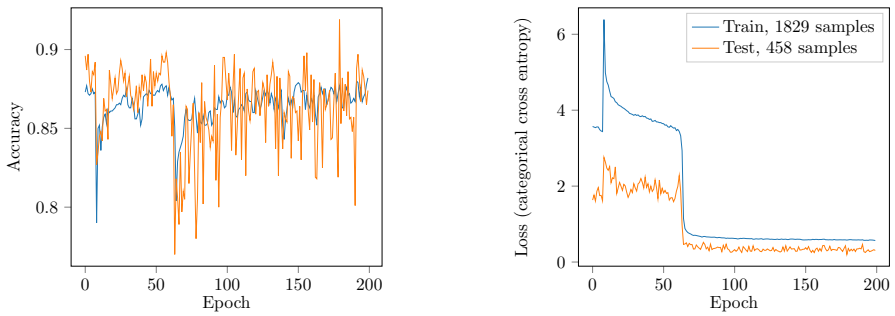


Figure 4.3: Model trained on the Swiss dataset

Figure 4.3 summarizes the results of our model trained during 200 epochs on the Swiss dataset. After the 179th epoch, our model achieved its highest accuracy (0.919) and lowest loss (0.20) on the test set. In Figure 4.4, the model was trained on the Massachusetts dataset during 140 epochs: The best results on the test set were achieved after the 133th epoch, with an accuracy and loss of 0.888 and 0.258, respectively. In both experiments, we notice that the test-accuracy oscillates a lot despite a low learning rate and decay. Using an even lower learning rate could solve this issue, but then the model's convergence speed would be much slower.

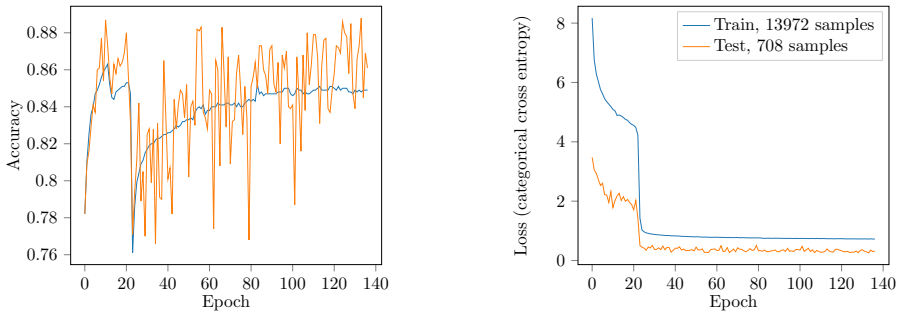
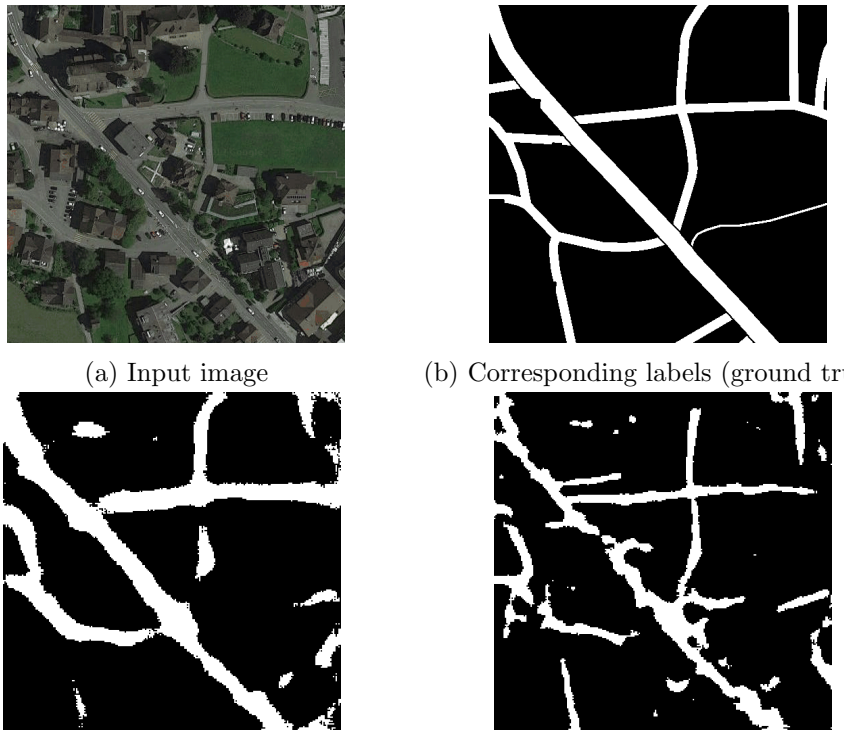


Figure 4.4: Model trained on the Massachusetts dataset

As explained above, the 'high' accuracies that we achieved should be taken with a grain of salt (because of extreme class imbalance) and thus, one also has to qualitatively assess the performances of the models. Moreover, we would like to know which dataset (Swiss or Massachusetts) is the best to train on. For this,



(a) Input image (b) Corresponding labels (ground truth)

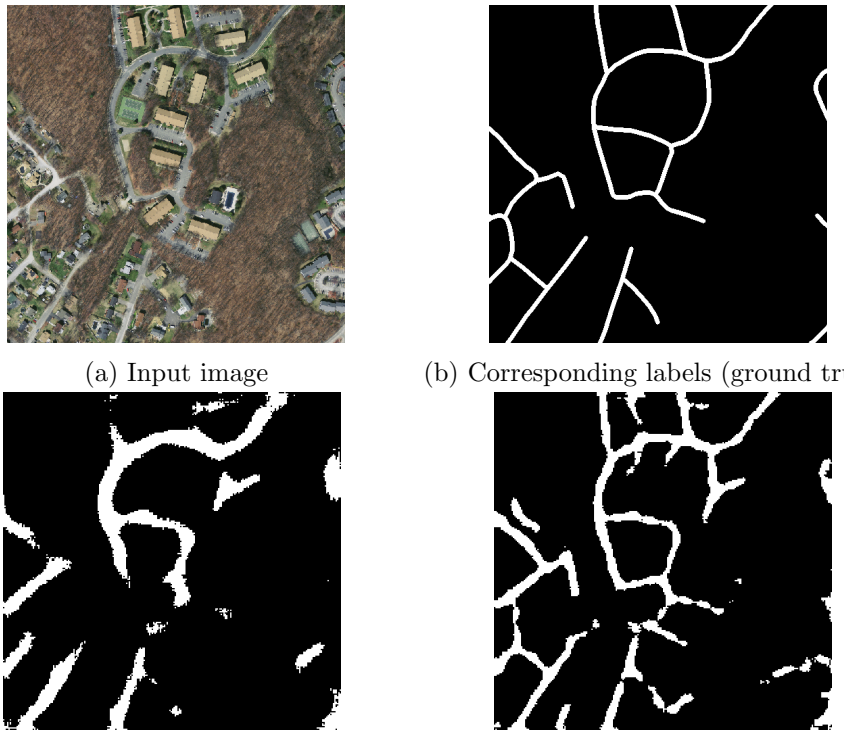
(c) Prediction, trained on Swiss data (d) Prediction, trained on Mass. data

Figure 4.5: Model's predictions on a test sample from the Swiss dataset

we perform the following experiment illustrated in the Figures 4.5 and 4.6: We take a test sample from the Swiss dataset and predict it twice, the first time using our model trained on the Swiss data, and the second time using our model trained on the Massachusetts data. We then do the same with a test sample from the Massachusetts data.

We can first see that both models perform qualitatively well, because in both cases roads seem to be detected and segmented quite accurately. However, the model trained on the Swiss dataset predicts roads that are quite larger than the ground truth (Figures 4.5c and 4.6c), which is not the case with the other model (Figures 4.5d and 4.5d). On the other hand, the predicted images by the model trained on the Swiss dataset seem to be less 'noisy', because the predicted roads appear to be more contiguous or less fragmented.

Eventually, we have used both models on various test samples, and have noticed that the model trained on the Massachusetts dataset is generally more precise and qualitatively better than the model trained on the Swiss dataset. We think that this is due to the quality of the satellite images and labels in the training set. In fact, the Swiss dataset was generated by the Google Maps API, which does not always give accurate label images: A road pixel at position



(a) Input image (b) Corresponding labels (ground truth)

(c) Prediction, trained on Swiss data (d) Prediction, trained on Mass. data

Figure 4.6: Model's predictions on a test sample from the Massachusetts (Mass.) dataset

(x, y) in the label image could be found at position $(x + t, y + t)$ (for a small displacement t) in the corresponding satellite image, which results in road pixels that are a bit shifted. On the other hand, the Massachusetts labels have been manually segmented, which generally results in data that are more faithful to the ground truth. Also, the Massachusetts training set contains more samples than the Swiss one (13972 samples versus 1829), which could also partially explain the differences in the predictions' quality. Table 4.1 shows a summary of our results for this chapter.

Model trained on	
Massachusetts dataset	0.919 at epoch 179
Swiss dataset	0.888 at epoch 133

Table 4.1: Best accuracy of our model on the test set for the road detection task.

4.1 Side Note: Building Detection

The network we developed for road detection can easily be adapted to any kind of semantic segmentation. Fortunately, the Massachusetts dataset ([6, 5]) not only contains data for roads, but also buildings in the satellite images, which can simply be fed to the same neural network we used for road segmentation. Figure 4.7 shows the prediction of our network when trained with this new dataset, with the corresponding ground-truth and satellite images.

(a) Prediction

(b) Ground truth

(c) Input satellite image

Figure 4.7: Model's prediction on a test sample from the Massachusetts Buildings dataset

We note that it is indeed quite straightforward to use the same convolutional neural network for different semantic segmentation tasks, because the only thing we changed was the labels data (buildings or roads) in the training set. However, such a straightforward change would likely not give optimal results, as shown in the prediction of buildings in Figure 4.7a. In fact, we can see that only the regions of the buildings are well detected, but not the individual buildings themselves: When we look at the predicted image, it seems that the network tries to detect buildings in the same way as it detects roads (i.e, with a contiguous line or region). We believe that this could be solved by fine-tuning the weights (or costs) associated with each class in the loss function.

Chapter 5

Conclusion and Future Work

In this work, we have tackled several Machine Learning problems. The prediction of population and number of houses (Chapter 3) was decent in the case of the classification task, but not the regression one. On the other hand, road segmentation (Chapter 4) was satisfactory for both datasets used (Massachusetts and Switzerland). Moreover, in Chapter 2, we have shown how to automatically generate a dataset of satellite images and corresponding labels. We think that this is an effective way to get a lot of data for semantic segmentation of satellite pictures, by noticing that the APIs mentioned could be used for more than just road segmentation.

In our opinion, there are many ways in which our performances could be improved. In Chapter 4, one could use a lot more data with the APIs, and augment them with rotations and scaling for example. A post-processing algorithm could also be implemented and even be learned as mentioned in [4]. Concerning the regression task in Chapter 3, a semantic segmentation of the buildings could be applied to pre-process the images; the buildings' labels would be generated in the same way as we did it for roads in Section 2.2.

Finally, once those tasks have been fully improved and tested, the task of housing price prediction would have to be considered again. In the future, we should examine whether or not the information predicted in the satellite picture of a house can be used to determine its price.

Bibliography

[1] Gress, B.: Using semi-parametric spatial autocorrelation models to improve hedonic housing price prediction. Department of Economics (2004)

[2] Ahmed, E., Moustafa, M.: House price estimation from visual and textual features. arXiv preprint arXiv:1609.08399 (2016)

[3] Limsombunchai, V.: House price prediction: hedonic price model vs. artificial neural network (2004)

[4] Delio Vicini, Matej Hamas, T.P.: Road extraction from aerial images

[5] Mnih, V.: Machine Learning for Aerial Image Labeling. PhD thesis, University of Toronto (2013)

[6] : Massachusetts Roads and Buildings Dataset. https://www.cs.toronto.edu/~vmnih/data/ Accessed: 2017-07-30.

[7] : Federal Statistical Office. https://www.bfs.admin.ch/bfs/en/home.html Accessed: 2017-07-30.

[8] : Statistische Ämter des Bundes und der Länder. http://www.statistik-portal.de/Statistik-Portal/ Accessed: 2017-07-30.

[9] : Google Static Maps API. https://developers.google.com/maps/documentation/static-maps/ Accessed: 2017-07-30.

[10] : OpenStreetMap Nominatim API. https://nominatim.openstreetmap.org/ Accessed: 2017-07-30.

[11] : Polyline Python Package. https://pypi.python.org/pypi/polyline/ Accessed: 2017-07-30.

[12] : Python. https://www.python.org/ Accessed: 2017-07-30.

[13] : Keras. https://keras.io/ Accessed: 2017-07-30.

[14] : Theano. http://deeplearning.net/software/theano/ Accessed: 2017-07-30.

[15] : Nvidia GeForce GTX 1080. https://www.nvidia.com/en-us/geforce/products/10series/geforce-gtx-1080/ Accessed: 2017-07-30.

[16] Zeiler, M.D.: Adadelta: an adaptive learning rate method. arXiv preprint arXiv:1212.5701 (2012)

[17] : Convolutional neural network (CNN). https://en.wikipedia.org/wiki/Convolutional_neural_network Accessed: 2017-08-16.

[18] Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556 (2014)

[19] Badrinarayanan, V., Kendall, A., Cipolla, R.: Segnet: A deep convolutional encoder-decoder architecture for image segmentation. arXiv preprint arXiv:1511.00561 (2015)

[20] Kingma, D., Ba, J.: Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980 (2014)