Notes to Image Processing and Modeling

Besides the metric and text-based features the Airbnb Data contains *images* of the listed apartments as well as of the corresponding hosts. This section describes how we used these images (while focusing on the apartment pictures) for the purpose of price prediction.

The main idea was to build a Convolutional Neural Network that predicts the price solely based on the image content itself and add these predictions as well as the number of available images for each listing to the main feature set. Since there exist multiple images per apartment, the predictions were averaged afterwards within each group to obtain an output array of equal length to all remaining features.

1 Webscraping

In the first step, the raw image links provided by the data set had to converted to an image format that the neural network is able to work with.

Therefore we first used the `requests` library to get the `HTML` Source Code of each listing's website. Next, the `beautifulsoup` library served as a convenient HTML parser to find and extract all embedded weblinks that lead to images located on the front page of the listings website. With this strategy we could extract the first 5 images for each apartment (if 5 or more were available) that could be directly accessed from the front page source code.

Finally, we used the `requests` module again in combination with the `pillow` package to decode the source content of all image adresses into two dimensional images.

2 Preprocessing

Before feeding these two dimensional images into the model, we performed some further preprocessing steps.

One very common technique when dealing with images is *Data Augmentation*. In contrast to classification tasks however, where Data Augmentation is used to expand the training set and improve simultaneously generalization, this approach is not immediately transferable to a regression context since we have to guarantee that the label (i.e. the price) remains unchanged for each image transformation. Thus, we decided against standard transformations such as rotating the images or manipulating the color composition.

We **did** use image *cropping* however which, in our opinion, is one of the few applicable augmentations in regression contexts. After resizing all images to 256×256 pixels we randomly cropped a square area of 224×224 out of each image in the training set and cropped an equally sized area out of the image **center** in the validation set to avoid any randomness during inference.

At a final step all images were normalized, separately for each color channel. Ic case of the pretrained model explained in the next section we used the same values for normilization that were used during training on the large `ImageNet` database. The mean values and standard deviations for each color channel are provided in the `PyTorch` documentation.

3 Modeling

As mentioned above, we used a pretrained Convolutional Neural Network for modeling. Ideally, due to learning from a large collection of labeled images in a supervised setting, this model is able to extract meaningful features from our own much smaller input data out of the box. As usual in *transfer learning* the weights of the pretraine model are frozen and the output layer is replaced by a trainable custom layer specific to our needs.

One potential issue arises if the dataset used for pretraining differs from the new custom data: If the pretrained network is very deep, the learned features before the final layer could be very specific to the Output Classes of `ImageNet` and not generalize well to our images.

There are multiple options to handle this scenario:

- Out of the vast collection of freely available pretrained models, choose one that is comparably shallow. We chose `ResNet18` with roughly 11 million parameters.
- Do not freeze the weights of the pretrained model completely, but rather fine tune them during training (i.e. modify *all* weights by backpropagating through the entire network). We did not investigate this option further due to its high computational cost.
- Cut the pretrained model before the last layer with the hope that, at this point, very generic and widely applicable features of images are extracted. These features might in theory generalize better to our data. In practice, however, this option did not improve our results significantly.

It turned out that a *single* custom layer, mapping from 512 directly to a single neuron representing the scalar price prediction, was not expressive enough. The performance improved by appending a (small) Fully Connected Network at the end instead containing three layers and `ReLU` activation functions.

To ensure that the chosen design is not majorly flawed, we constructed a separate much smaller Convolutional Neural Network with only a handful of Convolutional Blocks as a benchmark model. Although the performance differences were not as large as desired, the pretrained `ResNet` indeed indicated more promising results.

4 Results

Using only the content of the available images, the pretrained `ResNet18` achieved a Mean Absolute Error of 579 NOK (approx. 58 Euros) on the Validation Set. In comparison, the *Null Model* of always predicting the mean price achieved an MAE of 630 NOK without a log-transformation of the price and a MAE of 569 NOK with a log-transformation. Thus, the raw predictive power of the images alone was very small.

However, the *correlation* of the CNN predictions with the true price was 0.41. This indicates some limitations of the correlation as useful metric on the one hand but at least positive tendencies of the CNN predictions on the other hand. In fact, the network struggled the most with capturing the wide *range*

of prices and almost always predicted values close to the center of the (log) price distribution.

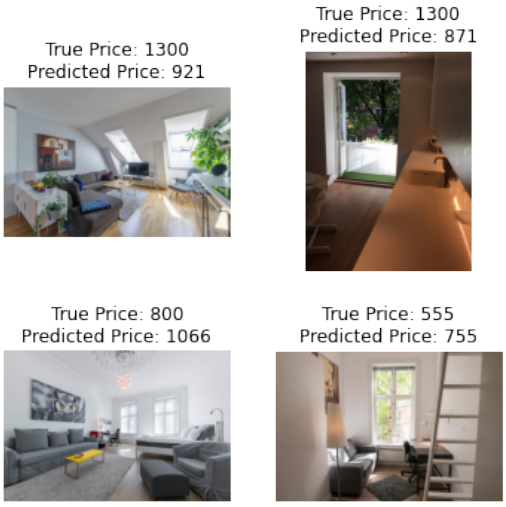

Figure 1: Images from Airbnb Apartments with true and predicted Prices

Although the general idea of categorizing images into price ranges based on image features sounds very appealing, taking a look at the actual input images reveals how challenging this task actually is. Figure 1 displays a random collection of input images. Considering the difficulty of the task it is actually highly doubtful that humans could provide much more accurate predictions.