

0. Abstract

1. Introduction

2. Research

2.1. Planning

There are many applications for the task to search for the shortest route on graph representation of map. We can abstract many real world scenario problems into this representation and then use many already existing algorithms commonly used for this class of tasks.

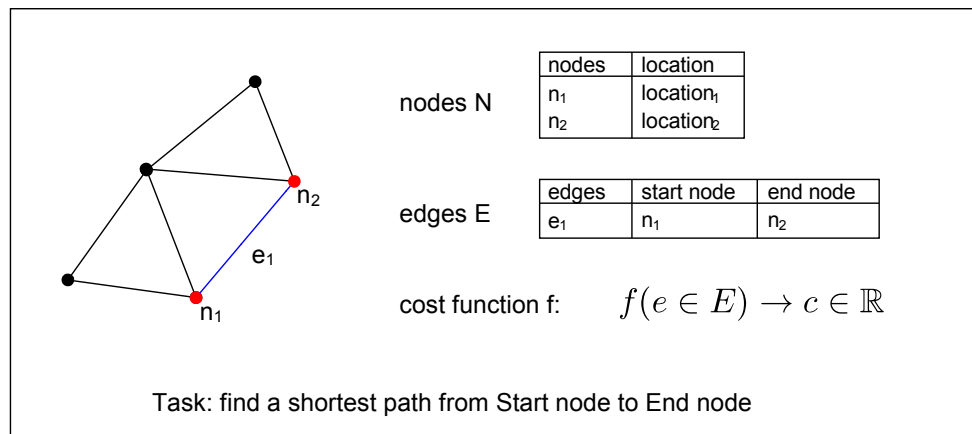


Figure 1: Abstracted map

We have a set of nodes N , which can be understood as places on map and edges E , which are the possible paths from one node to another, In order to have measurement of quality of traveling between two nodes, we also need a cost function. This cost function will assign a positive value $c \in \mathbb{R}_+$ to each edge. Typically we are facing the task of finding the shortest path, in which we are minimizing the aggregated cost. See illustration of [Figure 1].

In real life scenario, road segments exhibit many different parameters which influence how fast we can traverse them. More or less objective criteria such as surface material, size of the road, time of the day, or criteria which depends solely on the preference of driver. What is the surrounding environment, what is the comfort level of the road.

This tasks gets more interesting, when we look at more complicated examples, where the cost function is multi-criterial. In such case we need more data at our disposal and we can also expect the model to be more computationally difficult. For practical use, we need to use effective algorithm with speed up heuristics (see [hrnvcivr2016practical]). Great part of research is also in the area of hardware efficient algorithms, which would work on maps containing continent-sized datasets of nodes and edges, and yet coming up with solution in realistic time. We can expect such task on hand-held devices of car gps.

2.1.1. Data collection

Cos function can be very simple, but in order that it works on real life scenarios, we usually need more complicated one with lots of data recorded. Estimation of how much “cost” we associate

with one street (represented by edge $e \in E$ connecting two crossroads represented by nodes) should reflect how much time we spend in crossing it.

In case of planning for cars we generally just want to get across as fast as possible, or to cover minimizing distance. When the user is driving a bicycle, more factors become relevant. We need to know the quality of terrain, steepness of the road, amount of traffic in the area and overall pleasantness of the road. In many cases the bikers will not follow the strictly shortest path, choosing their own criteria, such as for example stopping for a rest in a park.

Some of these these criteria are measurable and objectively visible in the real world. For these we need to have highly detailed data available with parameters such as the quality of road and others. Other criteria are based on subjective, personal preference of some routes over other and for these we might need a long period of traces recording which routes have users selected in past. For example the work of [Navigation made personal] makes heavy use of user recorded traces.

See [Practical Multicrit.] and [Figure 2] for examples of types of measurements we would likely need to estimate cost of each edge considering the slowdown effect of these features.

surface	∈	[cobblestone, compacted, dirt, glass, gravel, ground, mud, paving stones, sand, unpaved, wood]
obstacle	∈	[elevator, steps, bump]
crossing	∈	[traffic signals, stop, uncontrolled, crossing]
cycleway	∈	[lane, shared busway, shared lane]
highway	∈	[living street, primary, secondary, tertiary]

Figure 2: Example of the categories of highly detailed data we would require for multi-criteria cost function formulation as presented by [Practical Multicrit.]. List of features contributing to a slowdown effect on route segment.

In any case highly qualitative, detailed and annotated dataset is required to begin with and a carefully fitted cost function which would take all these parameters into weighted account is also needed. Large companies are usually protective of their proprietary formulas of evaluating costs for route planners. For example [Navigation made personal] makes use of the road network data of Bing Maps with many parameters related to categories such as speed, delay on segment and turning to another street, however the exact representation remains unpublished.

As we will touch upon this topic in later chapters, its useful to realize that this highly qualitative dataset is not always available. We would like to carry information we can infer from small annotated dataset into different areas, where we lack detailed measurements. *(We are using visual information of Google Street View images, which is more readily available in certain areas than the highly qualitative dataset.)=if its now somewhere else already*

2.2. History of Convolutional Neural Networks

Initial idea to use Convolutional Neural Networks (CNNs) as model was introduced in [x] by LeCun with his LeNet network design trained on the task of handwritten digits recognition.

In this section we try to trace the important steps in the field of Computer Vision which lead to the widespread use of CNNs in current state of the art research.

2.2.1. ImageNet dataset

Computer Vision research has experienced a great boost in the work of [deng2009imagenet] in the form of image database ImageNet. ImageNet contains full resolution images built into the hierarchical structure of WordNet, database of synsets, “synonym sets” linked into a hierarchy.

WordNet is often used as a resource in the tasks of natural language processing, such as word sense disambiguation, spellcheck and other. ImageNet is a project which tries to populate the entries of WordNet with imagery representation of given synset with accurate and diverse enough images illustrating the object in various poses, viewing points and with changing occlusion.

As the work suggests, with internet and social media, the available data is plenty, but qualitative annotation is not, which is why such hierarchical dataset like ImageNet is needed. The argument for choosing WordNet is that the resulting structure of ImageNet is more diverse than any other related database. The goal is to populate the whole structure of WordNet with 500-1000 high quality images per synset, which would roughly total to 50 million images. Even till today, the ImageNet project is not yet finished, however many following articles already take advantage of this database with great benefits.

Effectively ImageNet became the huge qualitative dataset which was needed to properly teach large CNN models.

2.2.2. ImageNet Large Scale Visual Recognition Competition (ILSVRC)

The ImageNet Large-Scale Visual Recognition Challenge (ILSVRC) [Russakovsky2014] fueled the path of progress in the field of Image Recognition. While the tasks for each year slightly differs to encourage new novel approaches, it is considered the most prestigious competition in this field. The victorious strategies, methods and trends of each years of this competition offer a reliable looking glass into the state of art techniques. The success of these works and fast rate of progress has lead to popularization of CNNs into more practical implementations as is the case of Japanese farmer automatizing the sorting procedure of cucumbers [webpage of google blog post].

2.2.2.1. AlexNet, CNN using huge datasets

The task of object recognition has been waiting for larger databases with high quality of annotation to move from easier tasks done in relatively controlled environment, such as the MNIST task. In the work of [krizhevsky2012imagenet] the ImageNet database was used to teach a Deep Convolutional Neural Network (CNN) in a model later referred to as AlexNet. At the time this method has achieved more than 10% improvement over its competitors in an ILSVRC 2012 competition.

Given hardware limitations and limited time available for learning, this work made use of only subsets of the ImageNet database used in ILSVRC-2010 and ILSVRC-2012 competition. Choice of CNN as a machine learning model has been made with the reasoning that it behaves almost as well as fully connected neural networks, but the amount of parameters of connections is much smaller and the learning is then more efficient.

For the competition an architecture of five convolutional and three fully-connected layers composed of Rectified Linear Units (ReLU) as neuron models was used. Other alterations on the CNN architecture were also employed to combat overfitting, to fine-tune and increase score and reflect the limitations of hardware. Output of last fully-connected layer feeds to a softmax layer which produces a distribution over 1000 classes as a result of CNN.

The stochastic gradient descent was used for training the model for roughly 90 cycles through training set composed of 1.2 million of images, which took five to six days to train on two NVIDIA GTX 580 3GB GPUs.

2.2.2.2. VGG16, VGG19, going deeper

Following the successful use of CNNs in ILSVRC2012, the submissions of following years tried to tweak the parameters of CNN architecture. Approach chosen by [VGG16] stands out because of its success. It focused on increasing the depth of CNN while altering the structure of network. In their convolutional layers they chose to use very small convolution filter (with 3x3 receptive field), which leads to large decrease of amount of parameters generated by each layer.

This allowed them to build much deeper architectures and acquire second place in the classification task and first place in localization task of ILSVRC 2014.

2.2.2.3. ResNet, recurrent connections and residual learning

The work of [ResNet] introduced a new framework of deep residual learning which allowed them to go even deeper with their CNN models. They encountered the problem of degradation, where accuracy was in fact decreasing with deeper networks. This issue is not caused by overfitting as the error was increased both in the training and validation dataset.

Alternative architecture of model, where a identity shortcut connection was introduced between building blocks of the model, allowing it to combat this degradation issue and in fact gain better results with increasing CNN depth.

Their model ResNet 152 using 152 layers achieved a first place in the classification task of ILSVRC 2015.

2.2.2.4. Ensemble models

The state of the art models as of ILSVRC 2016 made use of the ensemble approach. Multiple models are used for the task and final ensemble model weights their contribution into an aggregated score.

The widespread of using the ensemble technique reflects the democratization and emergence of more platforms and public cloud computing solutions giving more processing power to the competing teams of ILSVRC.

2.2.3. Feature transfer

The success of large CNNs with many parameters trained on large datasets like ImageNet has not only been positive, it also poses a question – will we always need huge datasets like ImageNet to

properly teach CNN models? ImageNet has millions of annotated images and it has been gradually growing over time.

Article [Oquab2014] talks about this issue and proposes a strategy called feature transfer or model finetuning, which composes of using CNN models trained at one task to be retrained to a different task.

They offer a solution, where similar architecture of CNNs can be taught upon one task and then several of its layers can be reused, effectively transferring the mid-level feature representations to different tasks.

This can be used, when the source task has a rich annotated dataset available (for example ImageNet), whereas the target one doesn't. When talking about two tasks, the source and target classes might differ, when for example the labeling is different. Furthermore the whole domain of class labels can be also different – for example two datasets of images, where first mostly exhibits single objects and the second one rather contains more objects composed into scenes. This issue is referred to as a “dataset capture bias”.

The issue of different class labels is combated by adding two new adaptation layers which are retrained on the new set of classes. The problems of different positions and distributions of objects in image is addressed by employing a strategy of sliding window decomposition of the original image, continuing with sensible subsamples and finally having the result classifying all objects in the source image separately.

The article also works with a special target dataset Pascal VOC 2012 containing difficult class categories of activities described like “taking photos” or “playing instrument”. In this case the source dataset of ImageNet doesn't contain labels which could overlap with these activities, yet the result of this “action recognition” task achieves best average precision on this dataset.

This article gives us hope, that we can similarly transfer layers of Deep CNN trained on the ImageNet visual recognition source task to a different target task of Google Street View imagery analysis for cost estimation over each edge segment.

Articles being submitted to the ILSVRC competition often include a section dedicated on using their designed models on different tasks than what they were trained upon. Effectively they should the models suitability for feature transferring.

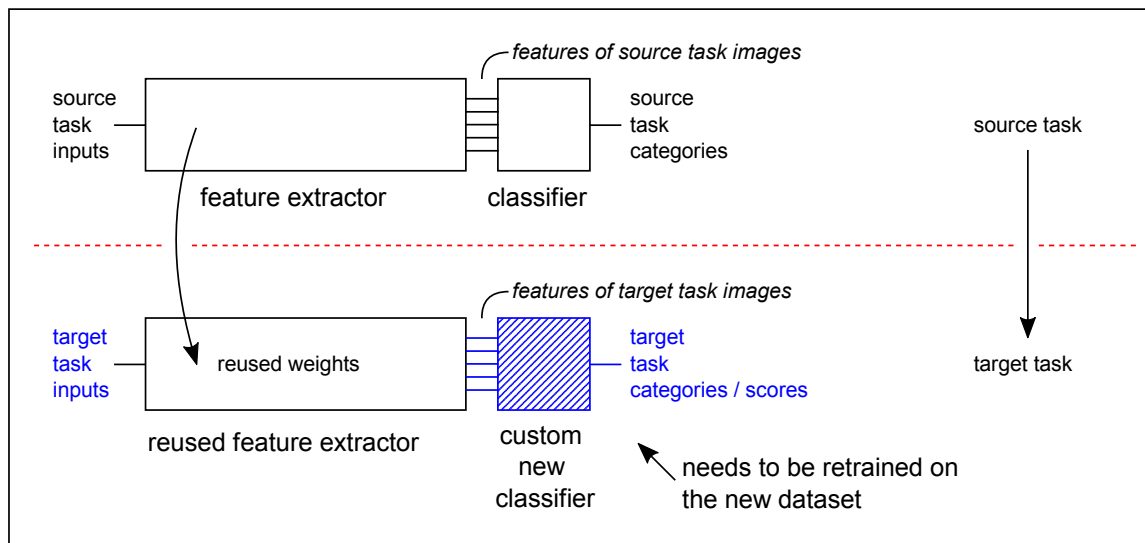


Figure 3: Basic idea of feature transfer between source and target task. Imagine the source task source task being classification task on ImageNet and the target task as a new problem without large dataset at its disposal.

2.2.4. Common structures

When designing the architecture of custom CNN model, we are using certain layers and building block schemes established as common practice in the ILSVRC competition. For a new unresearched task it has been suggested (by lecturers and online sources such as [website of the class]) to stick to an established way of designing the overall architecture.

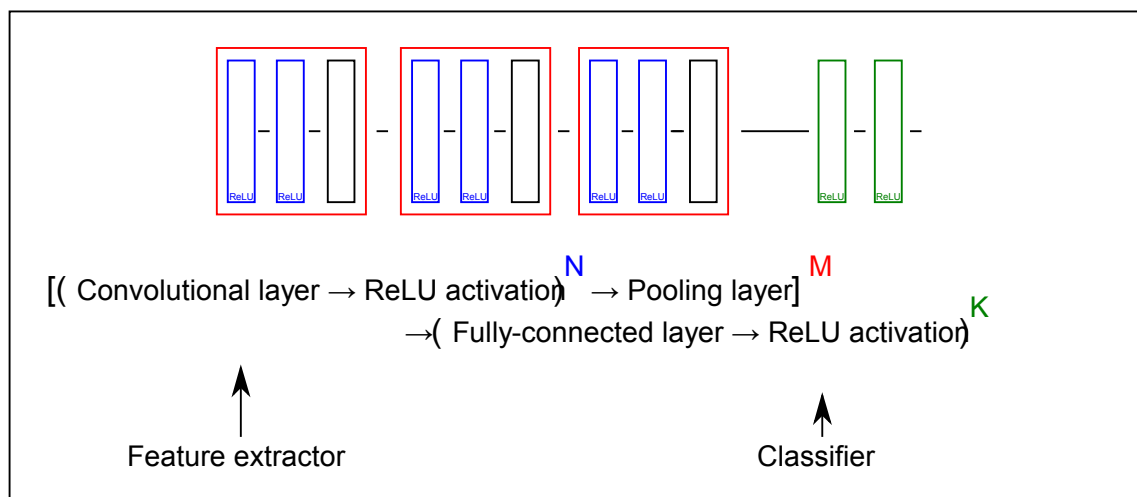


Figure 4: Structure

Refer to Figure 4. For illustration of this recommended architecture. Naturally for custom tasks this architecture is later adapted and tweaked to serve well in its specific situation. We will return to this suggested architecture scheme when building our own custom CNN models in [4.1. Building Blocks].

3. The Task

3.1. Route planning for bicycles

The task we are faced with consists of planning a route for bicycle on a map of nodes and edges. We are designing an evaluation method, which will give each edge segment appropriate cost. In such a way we are building one part of route planner, which will use our model for cost evaluation and fit into a larger scheme mentioned in [2.1. Planning].

As has been stated in [2.1.1. Data collection] a cost function can be an explicitly defined formula depending on many measured variables. Similar formula has been used by the ATG research group, which produced a partially annotated section of map with scores of bike attractivity. We want to enrich this dataset with additional visual information from Google Street View and with vector data of Open Street Map.

We want to train a model on the small annotated map segment and later use it in areas where such detailed information is not available. We argue that Google Street View and Open Street Map data are more readily obtainable, than supply of highly qualitative measurements.

3.2 Available data and collection

3.2.1. Initial dataset

We are given a dataset from the ATG research group of nodes and edges with score ranking ranging from 0 to 1. Score of 0 denotes, that in simulation this route segment was not used and value 1 means that it was a highly attractive road to take.

Each node is supplied with longitude, latitude location, which gives us the option to enrich them with additional real world information. Figure 5. shows the structure of initial data source.

Figure 5.

3.2.2 Street View

As each edge segment connecting two nodes is representing a real world street connecting two crossroads, we can get additional information from the location. We can download one or more images alongside the road and associate it with the edge and it's score from the initial dataset.

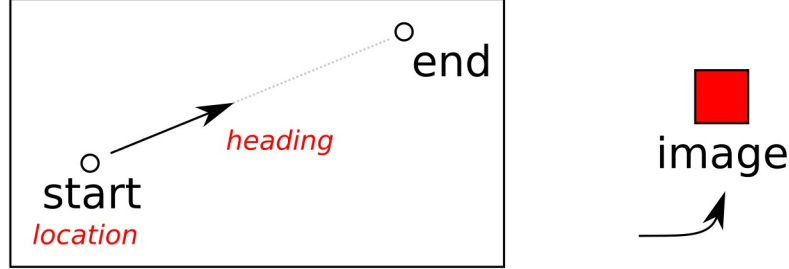
We are using a Google Street View API which allows us to generate links of images at specific locations and facing specific ways.

3.2.2.1. Downloading Street View images

Google Street View API uses the parameters of location which is the latitude and longitude and heading, which is the deviation angle from the North Pole in degrees. See Figure 6. In calculation of heading we are making a simplification of Earth being spherical using formula for initial bearing [formula 1].

$$\begin{aligned}
y &= \sin(lon_2 - lon_1) * \cos(lat_2) \\
x &= \cos(lat_1) * \sin(lat_2) - \sin(lat_1) * \cos(lat_2) * \cos(lon_2 - lon_1) \\
\Theta &= \text{atan2}(y, x)
\end{aligned}$$

Formula 1.



url: maps.googleapis.com/.../...&location=location&heading=heading

Figure 6.

In order to make good use of the location and collect enough data, we decided to break down longer edges into smaller segments maintaining the minimal edge size fixed. We also select both of the starting and ending locations of each segment. In each position we also rotate around the spot.

We collect total of 6 images from each segment, 3 in each of its corners while rotating 120° degrees around the spot. This allows us to get enough distinct images from each location and we don't overlap with neighboring edges. See the illustration in Figure 7. Note that all of these images will correspond to one edge and thus to one shared score value.

Also note that we are limited to downloading images of maximal size 640x640 pixels as per the limitation of free use of Google Street View API.

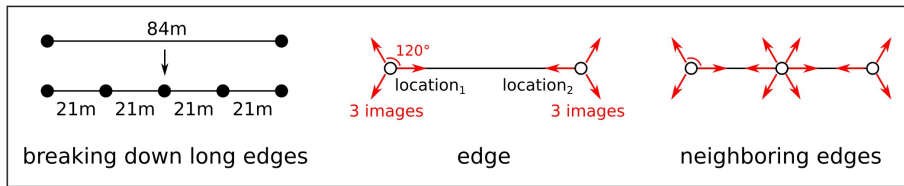


Figure 7.

3.2.2.2. Data augmentation introduction

In CNNs we often use the method of data augmentation to extend datasets by simple image transformations to overcome limitations of small datasets. We can generate crops of the original images, flip and rotate them or alter their colors.

We will return to the issue of dataset augmentation in [4.5. Data Augmentation]

3.2.3. Neighborhood in Open Street Map data

We are looking for another source of information about an edge segment in the neighborhood surrounding its location in Open Street Map data.

Open Street Map data is structured as vector objects placed with location parameters and large array of attributes and values. We can encounter point objects, line objects and polygon objects, which represent points of interest, streets and roads, building, parks and other landmarks.

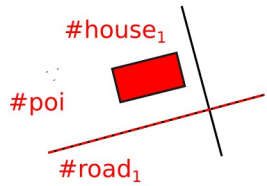
map sample	examples of parameters in format attribute=value
	<p>house₁: polyline object building=house, landuse=residential, ...</p> <p>road₁: line object higway=primary, surface=asphalt, ...</p>

Figure 8. / Example of structure of OSM data with parameters

From implementation standpoint, we have downloaded the OSM data covering map of our location and loaded it into a PostgreSQL database. In this way we can send queries for lists of objects in the vicinity of queried location. We will get into more detail about implementation in [5.4. OSM Marker].

3.2.3.1. OSM neighborhood vector

The structure of OSM data consists of geometrical objects with attributes describing their properties. In the PostgreSQL database each row represents object and attributes are kept as table columns.

Depending on the object type, different attributes will have sensible values while the rest will be empty. For better understanding consult table T. with examples of attributes and their values and table L. for examples of objects in OSM dataset.

attribute	possible values
highway	residential, service, track, primary, secondary, tertiary, ...
building	house, residential, garage, apartments, hut, industrial, ...
natural	tree, water, wood, scrub, wetland, coastline, tree_row, ...
surface	asphalt, unpaved, paved, ground, gravel, concrete, dirt, ...
landuse	residential, farmland, forest, grass, meadow, farmyard, ...

[Table T.]

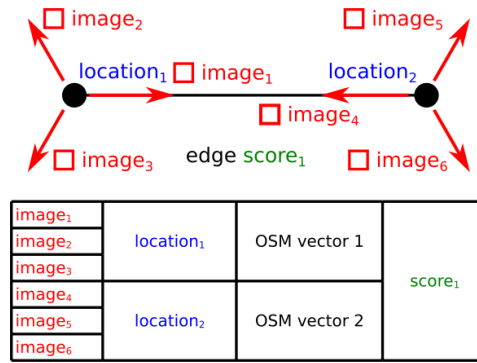


Figure F. / some same data in dataset because of how we generate images

One further undivided edge segment contains 6 images which share the same score, and some of which will share location and therefore also their OSM vectors. For illustration see Figure F.

3.2.3.2. Radius choice

Depending on the radius we choose different area will be considered as neighborhood. If we were to choose too small radius, the occurrences would mostly result in zero OSM vector. On the other hand selecting too high radius would lead many OSM vectors to be indistinguishable from each other as they would share the exact same values.

Our final choice for radius was value of 100 meters, which was empirically tested in [6. Results].

3.2.3.3. Data transformation

Similar to the spirit of data augmentation for images, we can try editing the OSM vectors in order that they will be more easily used by CNN models.

Instead of raw data of occurrences, we can convert this information into one-hot categorical representations or reduce them into Boolean values. Multiple readings of varying radius size can also be used for better insight of the neighborhood area.

See more about data augmentation in [4.5. Data Augmentation].

4. The Method

Our method will rely upon using Convolutional Neural Networks (CNNs) mentioned in [2.2. History of Convolutional Neural Networks] on an annotated dataset described in [3. The Task]. As we have enriched our original dataset with multiple types of data, particularly imagery Street View data and the neighborhood vectors, we have an option to build more or less sophisticated models, depending on which data will they be using. We can build a model which uses only relatively simple OSM data, or big dataset of images, or finally the combination of both.

Depending on which data we choose to use, different model architecture will be selected. Furthermore we can slightly modify each of these models to tweak its performance.

4.1. Building blocks

Regardless of the model type or purpose, there are certain construction blocks, which are repeated in the architecture used by most CNN models.

4.1.1. Model abstraction

When building a CNN model, we can observe an abstracted view of such model in terms of its design. Whereas at the input side of the model we want to extract general features from the dataset, at the output side we strive for a clear classification of the image.

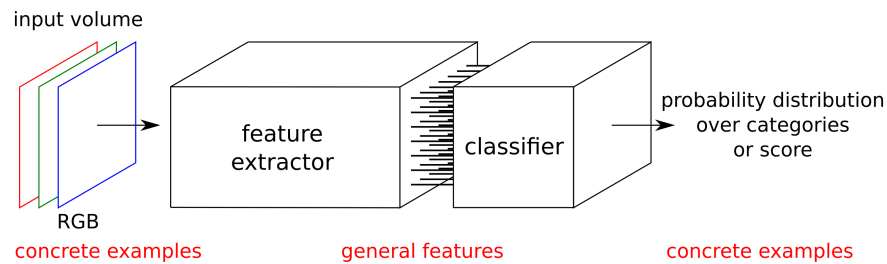


Figure G. / feature extractor + classification

Each of these segments will require different sets of building blocks and will prioritize different behavior. Good model design will lead to generalization of concrete task-specific data into general features, which will then be again converted into concrete categories or scores. The deeper the generic abstraction is, the better the model behavior in terms of overfitting will be.

Classification segment transforms the internal feature representation back into the realm of concrete data related to our task. In our tasks we are interested in score in range from 0 to 1 as illustrated by Figure H.

$$\text{output} = \text{sigmoid}(\text{dot}(\text{input}, \text{weights}) + \text{bias})$$

output $\in <0,1>$

parameters

The diagram shows a neuron with an input x_i and a weight w_i entering a summation node Σ . The output of the summation node is labeled 'activation sigmoid'.

Figure H. / output = sigmoid(dot(input,weights)+bias)

4.1.2. Fully-connected layers

The fully-connected layer denoted as “Dense” in Keras stands for a structure of neurons connected with every input and output by weighed connections. In Neural Networks these are named as hidden layers.

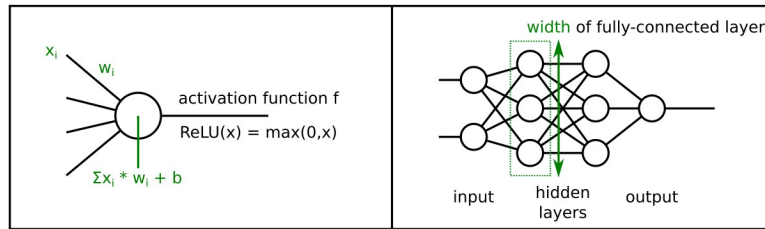


Figure CH. / input hidden layer output, width / Shows the model of connections of neurons in fully-connected layer.

The fully-connected layer suffers from a large amount of parameters it generates – weights in each connection between neurons and biases in individual neuron units. Fully-connected layers are usually present in the classification section of the model.

4.1.3. Convolutional layers

Convolutional layers are trying to circumvent the large amount of parameters of fully-connected layers by localized connectivity. Each neuron looks only at certain area of the previous layer.

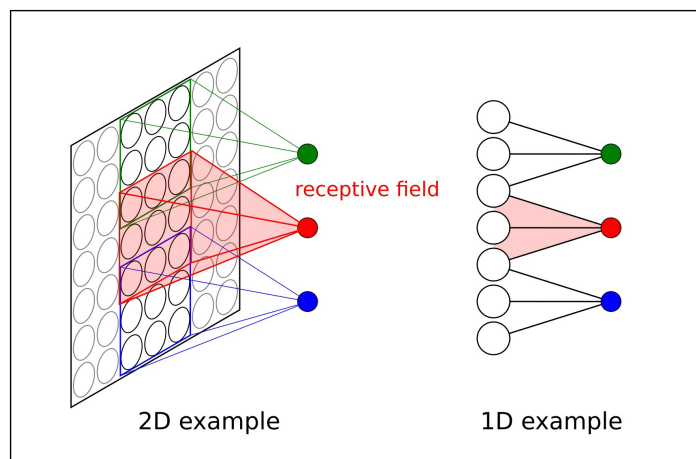


Figure I. / 2d 1d examples of convolutional layer connectivity

For their property to use considerably less parameters while at the same time to focus on features present in particular sections of image, they are often used as the main workforce in the feature extractor section of CNN models.

4.1.4. Pooling layers

Pooling layers are put in between convolutional layers in order to decrease the size of data effectively by downsampling the volume. This forces the model to reduce its number of parameters and to generalize better over the data. Pooling layers can apply different functions while they are downsampling the data – max, average, or some other type of normalization function.

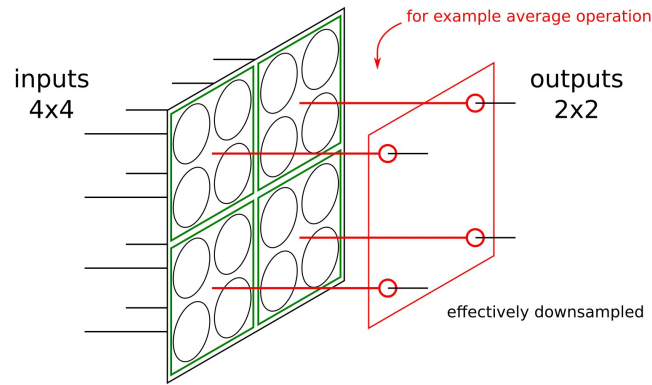


Figure J. pooling layer img

4.1.5. Dropout layers

Dropout layer is special layer suggested by [srivastava14a] which has since been widely used on the design of CNN architectures as a tool to prevent model overfitting.

The dropout layer placed between two fully-connected layers functions randomly drops connections between neurons with certain probability during the training period. Instead of fully-connected network of connections we are left with a thinned network with only some of the connections remaining. This thinned networks is used during training and prevents neurons to rely too much on co-adaptation. They are instead forced to develop more ways to fit the data as there is the effect of connection dropping. During test evaluation, the full model is used with its weights changed by the probability of dropout probability.

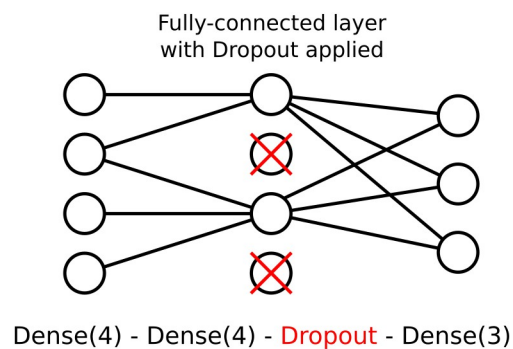


Figure K. dropout img

4.2. Open Street Map neighborhood vector model

In this version of model, we broke down edge segments formerly representing streets in real world into regularly sized sections each containing two locations of its beginning and ending location. These locations were enriched with OSM neighborhood vectors in [3.2.3. Neighborhood in Open Street Map data].

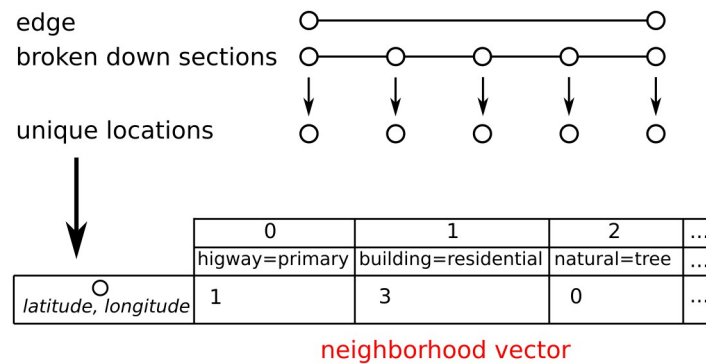


Figure L. / one edge to many to only locations + each distinct location attr=val table

Single unit of data is therefore a neighborhood vector linked to each distinct location of the original dataset. We have designed a model which takes these vectors as inputs and scores as outputs.

The OSM vector model is built from repeated building blocks of fully-connected layer followed by dropout layer. Fully connected layer of width 1 with sigmoid activation function is used as the final classification segment. Figure N. shows the model alongside its dimensions.

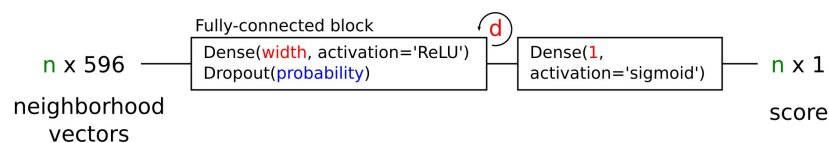


Figure N. / osm_only model

4.3. Street View images model

Each edge segment is represented by multiple images captured via the Street View API. Images generated from the same edge segment will share the same score label, however the individual images will differ. We can understand one image-score pair as a single unit of data.

The image data can be augmented in order to achieve richer dataset, see [4.5. Data Augmentation].

As discussed in [2.2. History of Convolutional Neural Networks] we are using a CNN model which has been trained on ImageNet dataset. We reuse parts of the original model keeping its weights and attach a new custom classification segment architecture at the top of the model.

We can generally divide even the more complicated CNN models into two abstract segments as mentioned in [4.1.1. Model abstraction]. The beginning of the model, which usually comprises of repeated structure of convolutional layers, followed by a classification section usually made of fully-connected layers.

The former works in extracting high dimensional features of incoming imagery data, whereas the classification section translates those features into a probability distribution over categories or

score. In our case we are considering a regression problem model, which works with score instead of categories.

As was mentioned in [2.2.3. Feature transfer] we reuse the model trained on large dataset of ImageNet, separate it from its classificatory and instead provide our own custom made top model. See Figure O.

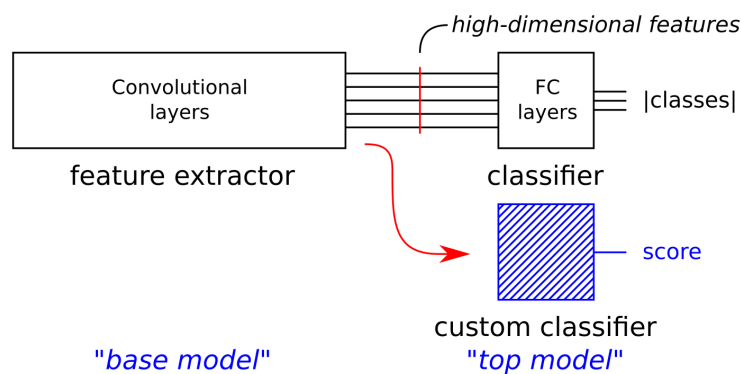


Figure O. / base model cut out of its own classifier

We prevent the layers of base model from changing their weights and train only the newly attached top model for the new task. There are certain specifics connected with this approach which we will explore in [4.6.3. Specific setting for models using images] section.

4.3.1. Model architecture

The final model architecture is determined by two major choices: which CNN to choose as its base model and how to design the custom top model so it's able to transfer the base models features to our task.

4.3.1.1. Base model

The framework we are working with, Keras, allows us to simply load many of the successful CNN models and by empirical experiments asses, which one is best suited for our task. More about Keras in the appropriate section [4.8.1. Keras].

The output of what remains of the base CNN model is data in feature space. The dimension will vary depending on the type of model we choose, the size of images we feed the base models and also the depth in which we chose to cut the base CNN model.

The remains of base CNN model are followed by a Flatten layer which converts the possibly multidimensional feature data into a one dimensional vector, which we can feed into the custom top model.

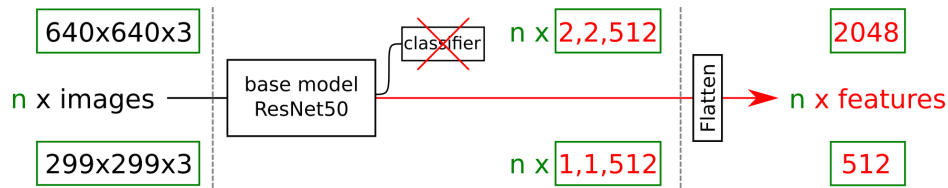


Figure P. / Example of differently sized images on the input and resulting feature vector size.

4.3.1.2. Custom top model

We feed the feature vector into a custom model built of repeated blocks of fully-connected neuron layers interlaced with dropout layers. The number of neurons used in each of the layers influences the so called model “width” and the number of used layers influences the model “depth”. Both of these attributes influence the amount of parameters of our model. We can try various combinations of these parameters to explore the models optimal shape.

The final layer of the classification section consists of fully-connected layer of width 1 with sigmoid activation function which weighs in all neurons of the previous layer. See Figure Q. Note that in the final model we chose to interlace individual fully-connected layers with dropout layers.

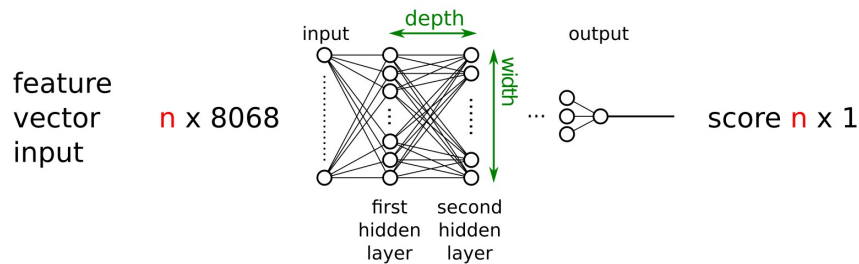


Figure Q. / feature input - first hidden - ... - output score, custom top model

4.3.1.3. The final architecture

The final architecture composes of base CNN model with its weights trained on the ImageNet dataset and of custom classification top model trained to fit the base model for our task.

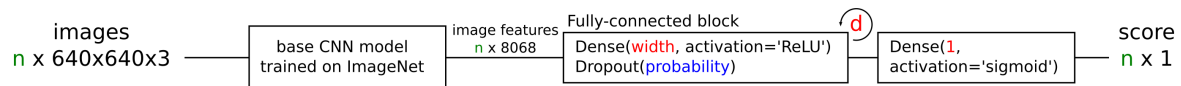


Figure R. / architecture of img model

4.4. Mixed model

After discussing the architecture of two models making only a partial use of the data collected in our dataset, we would like to propose a model combining the two previous ones.

In this case one segment again generates multiple images, which share the same score and depending on how they are created they could also share the same neighborhood OSM vector

representing the occurrences of interesting structures in its proximity. Different edges will generate not only different images, but also different scores and OSM vectors.

As a single unit of data we can consider the triplet of image, OSM vector and score. It's useful to note that later in designing the evaluation method of models, we should take into account, that the neighborhood vector and score can be repeated across data. When splitting the dataset into training and validation sets, we should be careful and place images from one edge into only one of these sets. Otherwise data with distinct images, but possibly the same neighborhood vector and score could end up in both of these sets. Figure S. illustrates how single data units are generated from one edge.

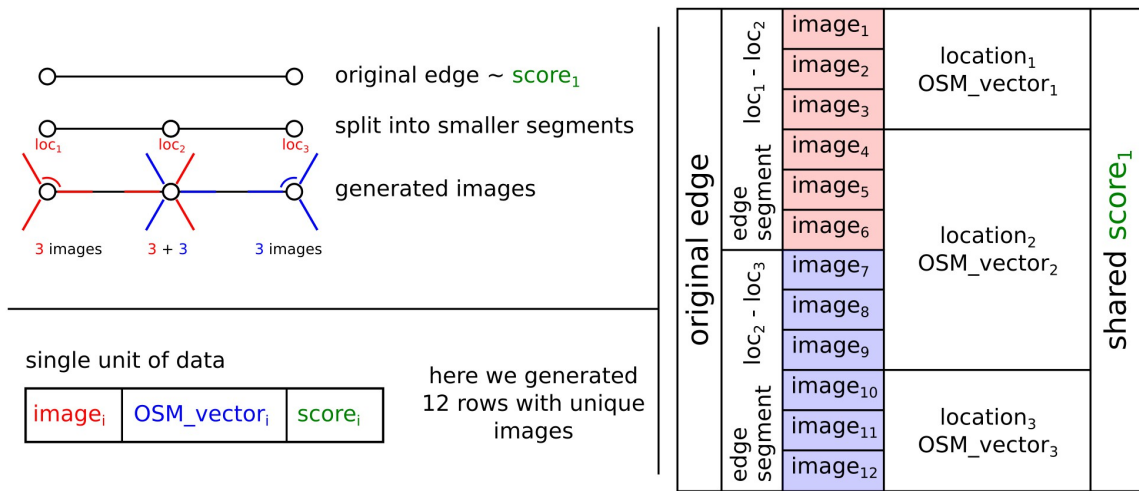


Figure S. / Example of possible decomposition of original edge into rows of data accepted by the mixed model.

We can join the architectures designed in previous steps, or we can design a new model. We chose to join the models in their classification segment. We propose a basic idea for a simple model architecture, which concatenates feature vectors obtained in previous models and follows with structure of repeated fully-connected layers with dropout layers in between. Concatenation joins the two differently sized one dimensional vectors into one. As is observable on Figure T. we use several parameters to describe the models width and depth.

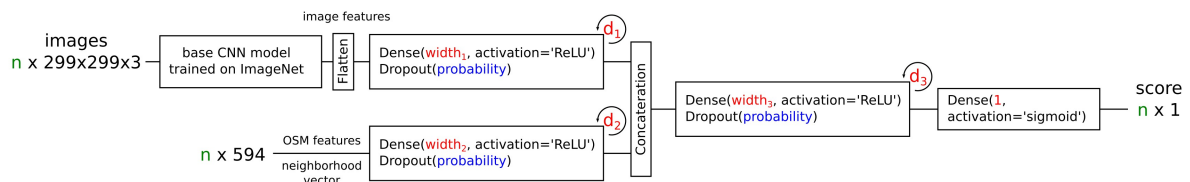


Figure T. / final architecture of mixed model

4.5. Data Augmentation

4.6. Model Training

4.6.1. Dataset split into validation and training data

- 4.6.2. Training setting
- 4.6.3. Specific setting for models using images
- 4.6.4. Feature cooking
- 4.7. Model evaluation
 - 4.7.1. K-fold cross validation
- 4.8. Frameworks and projects
 - 4.8.1. Keras
 - 4.8.2. Metacentrum project
- 5. The Implementation
 - 5.1. Project overview
 - 5.2. Experiment running
 - 5.3. Downloader functionality
 - 5.4. OSM Marker
 - 5.5. Datasets and DatasetHandler
 - 5.5.1. Dataset statistics
 - 5.5.2. DatasetHandler structure
 - 5.6. Models and ModelHandler
 - 5.6.1. Model description in Keras
 - 5.6.2. ModelHandler structure
 - 5.7. Settings structure
 - 5.8. Training
 - 5.9. Testing
 - 5.10. Reporting and folder structure
 - 5.11. Metacentrum scripting
- 6. Results
- 7. Discussion
- 8. Conclusions