DeepNav: Learning to Navigate Large Cities

Samarth Brahmbhatt
Georgia Institute of Technology
Atlanta USA
samarth.robo@gatech.edu

James Hays
Georgia Institute of Technology
Atlanta USA
hays@gatech.edu

Abstract

We present DeepNav, a Convolutional Neural Network (CNN) based algorithm for navigating large cities using locally visible street-view images. The DeepNav agent learns to reach its destination quickly by making the correct navigation decisions at intersections. We collect a large-scale dataset of street-view images organized in a graph where nodes are connected by roads. This dataset contains 10 city graphs and more than 1 million street-view images. We propose 3 supervised learning approaches for the navigation task and show how A search in the city graph can be used to generate supervision for the learning. Our annotation process is fully automated using publicly available mapping services and requires no human input. We evaluate the proposed DeepNav models on 4 held-out cities for navigating to 5 different types of destinations. Our algorithms outperform previous work that uses hand-crafted features and Support Vector Regression (SVR) [19].*

1. Introduction

Man-made environments like houses, buildings, neighborhoods and cities have a *structure* - microwaves are found in kitchens, restrooms are usually situated in the corners of buildings, and restaurants are found in specific kinds of commercial areas. This structure is also *shared* across environments - for example, most cities will have restaurants in their business district. It has been a long-standing goal of computer vision to learn this structure and use it to guide exploration of unknown man-made environments like unseen cities, buildings and houses. Knowledge of such structure can be used to recognize tougher visual concepts like occluded or small objects [8, 29], to better delimit the boundaries of objects [6, 38], and for robotic automation tasks like deciding drivable terrain for robots [13], etc.

In this paper, we address a task that requires understanding the large-scale structure of cities – street-view based navigation in a new city to reach a destination in as few steps as possible. The agent neither has a map of the environment, nor does it know the location of the destination or itself. All it knows is that it needs to reach a particular type of destination, e.g. go to the nearest gas station in the city. The naive approach is a random walk of the environment. But if the agent has some learnt model of the structure of cities, then it can make *informed decisions* - for example, gas stations are very likely to be found near freeway exits.

Figure 1: These street-view images are taken from roughly the same location. Which direction do you think the nearest gas station is in?[1]

We finely discretize the city into a grid of locations connected by roads. At each location, the agent has access only to street view images pointing towards the navigable directions and it has to pick the next direction to take a step in (Figure 1). We show that learning structural characteristics of cities can help the agent reach a destination faster than random walk. This technology can be used to guide pedestrians and cars in GPS denied environments like remote places or urban areas with tall buildings. A similar agent trained to navigate buildings to reach elevators, restrooms and fire exits can be used to guide visually impaired persons in unknown buildings.

[1] The correct answer is: W.

Two approaches have been used in the past to achieve this: 1) Reinforcement Learning (RL): a positive reward is associated with each destination location and a negative reward with all other locations. The agent then learns a policy (mapping from state to action) that maximizes the reward expected by executing that policy. In deep RL the policy is encoded by a CNN that outputs the value of performing an action in the current state. A state transition and the observed reward forms a training data-point. A mini-batch for CNN training is formed by some transitions that are driven by the current policy and some that are random. Recent works have used this approach to navigate mazes [27] and small game environments [30]. 2) Supervised Learning: this form of learning requires a large training set of images with labels (e.g. optimal action or distance to nearest destination) that would lead the agent to choose the correct navigable direction at the locations of the images.

Advances in deep CNNs have made it possible to learn high-quality features from images that can be used for various tasks. Most research is focused on recognizing the content of the images e.g. object detection [11, 20, 31], semantic segmentation [7, 25, 38], edge detection [23], salient area segmentation [22], etc. However, in robotics and AI one is often required to map image(s) to the choice of an action that a robotic agent must perform to complete a task. For example, the navigation task discussed in this paper requires the CNN to predict the direction of the next step taken by the agent. Such tasks require the CNN to assess the *future implications* of the content of an image, and are relatively unexplored in the literature.

We choose supervised learning for this task because of the sparsity of rewards. Out of roughly 100,000 locations in a city grid, only around 30 are destinations (sources of positive reward). RL needs thousands of iterations to sample a transition that ends at a destination (the only time a positive reward happens), especially early in the training process when RL is mostly sampling random transitions. Indeed, Mnih et al. [27] use 200M training frames to learn navigation in a small synthetic labyrinth. It is not clear if this approach will scale to large-scale environments with highly sparse rewards such as city-scale navigation. On the other hand, there exists an oracle for the problem of navigation in a grid: A* search. A* search finds the shortest path from a starting location to the destination, which gives the optimal action that the agent must perform at every location along the path. Hence it can be used for efficient labelling of large amounts of training images.

To summarize, our contributions are:

- We collect a dataset of roughly 1M street view images spread across 10 large cities in the USA. This dataset is marked automatically with locations of five types of destinations (Bank of America, church, gas station, high school and McDonald's) using publicly available mapping APIs [2, 3]
- We develop and evaluate 3 different CNN architectures that allow an agent to pick a direction at each location to reach the nearest destination. We also compare the performance of these 3 CNN-based models with the model described in [19], which used hand-crafted features and support vector regression for the same task.
- We develop a mechanism that uses A* search to generate appropriate labels for our architectures for all the images in the dataset.

The rest of the paper is organized as follows: §2 describes the related work in this area, §3 describes our dataset collection process, §4 describes the CNN architectures and training processes and §5 presents results from our algorithm. We discuss the results and conclude in §6.

2. Related Work

The computer vision community has explored scene understanding from the perspective of scene classification [20, 39], attribute prediction [24, 32, 39], geometry prediction [15] and pixel level semantic segmentation [7, 25, 38]. All these approaches, however, only reason about information directly present in the scene. Navigating to the nearest destination requires not only understanding the local scene, but also predicting quantities *beyond* the visible scene e.g. distance to nearest destination establishment [19]. Khosla et al. [19] is closest to our paper and addresses the task of navigating to the nearest McDonald's establishment using street-view images. They use a dictionary of spatially pooled Histogram of Oriented Gradient features [9] to learn a Support Vector Regressor [34] that predicts the distance to the nearest destination in the direction pointed to by the image. In this paper, we first use a CNN to predict the distance and show that data-driven convolutional features perform better than expensive hand crafted features for this task. Next, we propose 2 novel mechanisms to supervise a CNN for this task and show that they lead to better performance. A related line of work deals with image geo-localization [14, 37] - the problem of localizing the input image in a map. Kendall et al. [18] use CNNs to directly map an input image to the 6D pose of the camera that took the image, in a city-level environment. However, these algorithms only partially solve the problem addressed in this paper - the next steps involve determining the location of the destination and planning a path to it using a map.

In artificial intelligence, the problem of picking the optimal action by observing the local surroundings has recently been studied as an application of Deep Reinforcement Learning [28]. Works such as [27, 30, 40] use Deep RL to learn navigation in artificially generated labyrinths and Minecraft environments. However, the environments are much smaller than the city-level environments considered in this paper and have either repeating artificial pat-

Figure 2: Directed graph illustration. The red arrows represent nodes, encoded by location and direction. Each node has an associated street-view image, taken at the node's location and pointing in the node's direction. Nodes are connected by roads (solid connectors in the figure).

City	Images	BofA	church	gas station	high school	McDonald's
Atlanta	78,808	10	32	32	7	7
Boston	105,000	40	40	39	20	20
Chicago	105,001	22	33	10	15	32
Dallas	105,000	7	25	35	9	13
Houston	117,297	8	19	30	4	14
Los Angeles	80,701	9	15	30	6	13
New York	105,148	30	20	21	27	31
Philadelphia	105,000	14	42	35	30	19
Phoenix	101,419	4	23	29	18	15
San Francisco	101,741	35	50	45	22	12
Total	**1,005,115**	**179**	**299**	**306**	**158**	**176**

Table 1: Number of images and destinations in city graphs

terns [27] or monotonous non-realistic video game renderings [30]. Another issue with Deep RL is the amount of training data and training epochs required. Even in small-scale environments with denser reward-yielding locations compared to our environments, Mnih et al. [27] use 50 training epochs with each epoch made of 4M training frames, while Oh et al. [30] use up to 200 epochs. In contrast, our algorithms require 8 epochs to train, while our initialization network [33] requires 74 training epochs through 1M images. To our knowledge, no deep RL algorithm has attempted the task of learning to navigate in city-scale environments with real-world noisy imagery. Mirowski et al. [26] use auxiliary tasks like depth prediction from RGB and loop closure detection to alleviate the sparse rewards problem. However, their experiments are performed in artificially generated environments much smaller than the city-scale environments we operate in.

In robotics, active vision [5] has been used to control a robot to reach a destination based on local observations. The use of shortest paths to train classifiers that map the state of an agent (encoded by local observations) to action first appears in [17]. They train a decision tree on various hand-crafted attributes of locations in a supermarket (e.g. type of aisle, visible products, etc.) to control a robot to reach a target product efficiently. Aydemir et al. [4] use a chain graph model that relies on object-object co-occurrence and object-room type co-occurrence to control a robot to reach a destination in a 3D indoor environment.

3. Dataset

The DeepNav agent has a location and a heading associated with it, and traverses a directed graph covering the city. Figure 2 shows a visualization of a small part of this graph. Nodes are defined by the tuple of street view image location (latitude, longitude) and direction (North, South, East, and West). Hence each location can host upto 4 nodes. Edges in this graph represent a one-way road i.e. a node is connected to a neighbouring node by a directed edge if there is a road that allows the agent to travel from the first node to the second node. However, edges only connect neighbouring nodes facing the same direction. Hence travelling along an edge allows the agent to take one step in the direction of its current heading. To allow the agent to turn in place, all nodes at one location are cyclically connected with bidirectional edges. Lastly, a node exists only if it is connected to a node at a different location. This implies that locations along a road only have 2 nodes per location, while intersections have 3 or 4 nodes per location depending on the type of intersection. At intersections of more than 4 roads, we ignore all roads that do not point in the cardinal directions. This construction makes it possible for the agent to travel between any two nodes in the graph. Each node has a 640 x 480 image cropped from the Google Street View panorama at the location of the node. This image has a field-of-view of 90° and points in the direction associated with the node. While cardinal directions N, S, E, W are shorthand, the street-view crops have continuous direction consistent with the road direction.

To control the granularity of node locations, we tessellate the city limits into square bins of side 25m and consider the centers of these bins to be node locations. All street-view panorama locations that fall inside a bin are snapped to the center of the bin. However, the actual images are captured from the edges of the bins, to ensure visual continuity.

We generate one such graph per city. First the limits are specified by the latitude and longitude of two opposite corners of a rectangular region. We then start a breadth-first enumeration of the locations in the city, starting at the center of the rectangle (shown in the supplementary material). This enumeration stops when the specified geographical limits are reached. Table 1 shows the number of images in the graphs of the 10 cities in our dataset.

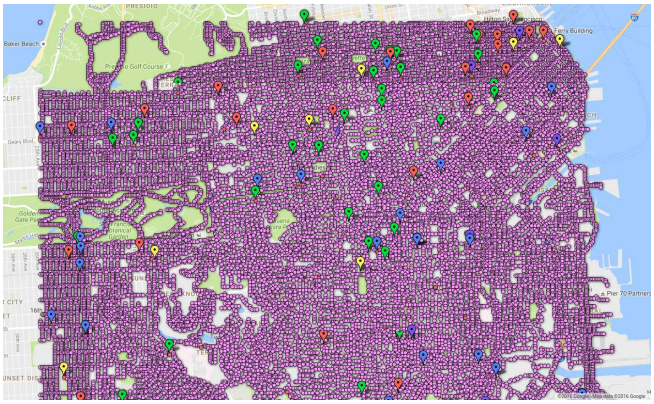


Figure 3: San Francisco graph locations and destinations (red: Bank of America, green: church, blue: gas station, yellow: high school, purple: McDonald's)

3.1. Destinations

We consider 5 classes of destinations: Bank of America, church, gas station, high school and McDonald's. These were chosen because of their ubiquity and distinguished appearance. Given the city limits we use Google Maps nearby search [3] with an appropriate radius to find the locations of all establishments of these classes. Next, we use the Google Maps Roads API [2] to snap these locations to the nearest road location. This is necessary because these establishments are often large in size and street-view images exist only along roads. Figure 3 shows the destination locations for San Francisco, while Table 1 shows the number of destinations found by our program in each city graph.

4. CNN Architecture and Training

Given the city graph and street-view images, we want to train a convolutional neural network to learn the visual features that are common across paths leading to various destinations. We propose 3 methods to label the training images (and corresponding inference algorithms) to accomplish this. The first approach, DeepNav-distance, trains the network to estimate the distance to the nearest destination in the direction pointed to by the training image. The second approach, DeepNav-direction, learns a mapping between a training image and the optimal action to be performed at the image location. The third approach, DeepNav-pair, decomposes the problem of picking the optimal action into pairs of decisions, and employs a Siamese CNN architecture.

4.1. DeepNav-distance

In this scheme we label each image with the square-root of the straight-line distance from the image location to the nearest destination establishment, in the 90° arc corresponding to the direction of the image. We collect 5 labels corresponding to 5 destination classes for each image. To train, we modify the last fully connected layer (fc8) of the VGG 16-layer network [33] to have 5 output units (see Figure 4a). The objective function minimized by this algorithm is the Euclidean distance between the fc8 output and the 5-element label vector. If a particular node has no destination establishment of a category in its arc, the corresponding element of the label vector is set to a high value that is ignored by the objective function. Hence, a training image is used for learning as long as it has at least one kind of destination in its arc.

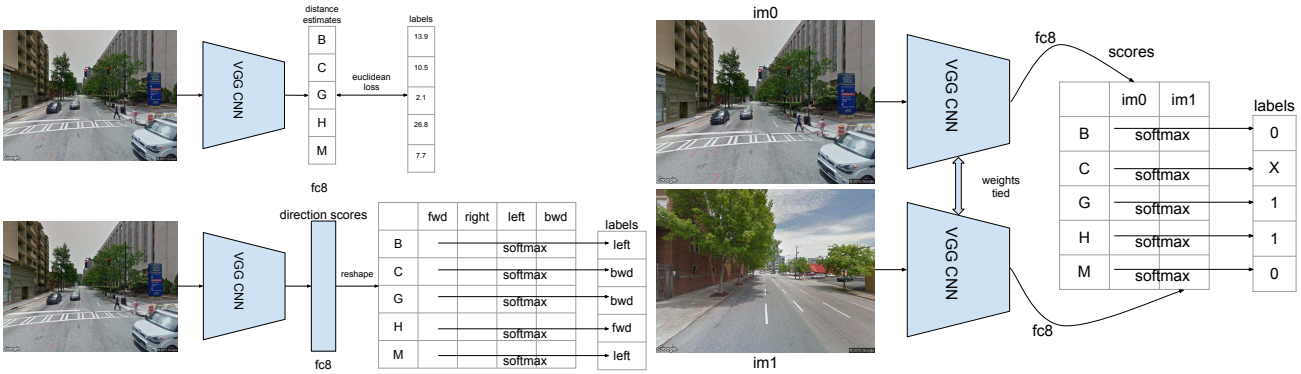
We use a greedy approach at test time: we forward images from all available directions at the current location of the agent through the CNN. The agent takes a step in the direction that is predicted by the CNN to have the least distance estimate. This approach is inspired from [19], and is intended to investigate the change in performance by using an end-to-end convolutional neural network pipeline instead of hand-crafted features and support vector regression.

4.2. DeepNav-direction

This approach learns to map an input image to the optimal action to be performed at that particular location and direction. The graph allows the agent to perform up to 4 actions at a node: move forward, move backward, move left or move right (the last 3 are composed of the primitive actions of moving forward and turning in place). We note that A* search in the graph finds the shortest path from any starting location to a destination, and hence can generate optimal action labels for each node location along the shortest path. For example, if the A* path at a node turns East, the image at that node facing East is labelled 'move forward', the image facing North is labelled 'move right', and so on. Algorithm 1 describes the process of generating labels for all training images using A* search, for one class of destination (e.g. high schools). The algorithm is repeated for all 5 classes of destinations to get 5 optimal action labels for each training image. Each label can take one of four values. To train, we modify the last fully connected layer (fc8) of the VGG 16-layer network [33] to have 20 output units (see Figure 4a). These 20 outputs are interpreted as scores for the 4 possible actions (along the columns) for the 5 destination classes (along the rows). The objective function minimized by DeepNav-direction is the softmax loss computed independently for each destination class. At test time, the image from the current position and direction of the agent is forwarded through the convolutional neural network, and the agent performs the highest scoring available action.

4.3. DeepNav-pair

This approach also learns to select the optimal action to be performed at a particular location and direction like DeepNav-direction, but through a different formulation.



(a) DeepNav-distance (top), DeepNav-direction (bottom)

(b) DeepNav-pair

Figure 4: DeepNav CNN architectures. Abbreviations for destinations: B = Bank of America, C = Church, G = Gas station, H = High school, M = McDonald's.

Data: City graph G, destinations D
Result: Optimal action labels for each node
while $\exists$ *unlabelled node* **do**
  $n \leftarrow$ unlabelled node;
  shortest_path $\leftarrow []$;
  min_cost $\leftarrow \infty$;
  foreach $d \in D$ **do**
    $cost, path \leftarrow \text{A}*(n, d, G)$;
    if $cost <$ min_cost **then**
      min_cost $\leftarrow cost$;
      shortest_path $\leftarrow path$;
    end
  end
  foreach *node* $i \in$ shortest_path **do**
    label each node at $i.location$ using A* action;
  end
end

Algorithm 1: Generating labels for DeepNav-direction

DeepNav-direction gets only the forward-facing image as input, and does not get to 'see' in all directions before choosing an action. This is an important action performed by a variety of animals including primates, birds and fish while navigating unknown environments [36]. This behaviour can be implemented by the Siamese architecture shown in Figure 4b. We enumerate all pairs of images at a location, and use the optimal action given by A* to label at most one image from each pair as the 'favorable' image. A pair is ignored if it does not contain a favorable image. For example, if the A* path at a node turns East, the second image in the North-East pair is marked favorable, while the North-South pair is ignored. Algorithm 2 shows the process of gathering the labels for all such pairs in the training dataset, and it is repeated for each destination class. To train, we create a Siamese network with 2 copies

Data: City graph G, destinations D
Result: Optimal action labels for each image-pair, where pairs are formed between images at a common location
while $\exists$ *unlabelled node* **do**
  $n \leftarrow$ unlabelled node;
  shortest_path $\leftarrow []$;
  min_cost $\leftarrow \infty$;
  foreach $d \in D$ **do**
    $cost, path \leftarrow \text{A}*(n, d, G)$;
    if $cost <$ min_cost **then**
      min_cost $\leftarrow cost$;
      shortest_path $\leftarrow path$;
    end
  end
  foreach *node* $i \in$ shortest_path **do**
    foreach *pair* $p \in pairs(i.location)$ **do**
      if $direction(p.first) == A*\ direction$ **then**
        $(p.first, p.second) \leftarrow$ label 0;
      else if $direction(p.second) == A*\ direction$ **then**
        $(p.first, p.second) \leftarrow$ label 1;
      else ignore pair
        $(p.first, p.second) \leftarrow$ label X;
      end
    end
  end
end

Algorithm 2: Generating labels for DeepNav-pair

of the DeepNav-distance network as shown in Figure 4b. The outputs of the fc8 layer are treated as scores instead of

distance estimates, and stacked as columns. A softmax loss is applied across the columns, independently for each destination. Hence the network learns to pick the image pointing in the optimal direction, from all existing images at a location. At test time we keep only one branch of the Siamese architecture and use the fc8 outputs as a score. All images from the current location of the agent are forwarded through the network, and the agent takes a step in the direction of the image that has the highest score predicted by the network.

4.4. Training

We train the convolutional neural networks using stochastic gradient descent (SGD) implemented in the Caffe [16] library. The learning rate for SGD starts at 10^{-3} for DeepNav-pair and DeepNav-direction, and 10^{-4} for DeepNav-distance. All models are trained for 8 epochs, with the learning rate dropping by a factor of 10 after the 4th and 6th epochs. We set the weight decay parameter to $5 * 10^{-4}$ and SGD momentum to 0.9. Training on an NVIDIA TITAN X GPU takes roughly 72 hours for each DeepNav model. All networks are initialized from the public VGG 16-layer network [33] except the fc8 layers, which are initialized using the Xavier [12] method.

4.4.1 Geographically weighted loss function

The DeepNav models are being trained to identify visual features that indicate the path to a destination. We expect these visual features to be concentrated around destination locations. To relax the CNN loss function for making a wrong decision based on low-information visual features far away from the destinations, we modify the loss function by weighing the training samples geographically. Specifically, the weight for the training sample reduces as length of the shortest path from its location to a destination increases. The geographically weighted loss function L_g for a mini-batch of size N is constructed from the original loss function L as $L_g = \sum_{i=1}^{N} \lambda^{l_i} L_i$, where $0 < \lambda < 1$ is the geographic weighting factor. We apply geographic weighting to the loss functions for DeepNav-direction and DeepNav-pair with $\lambda = 0.9$. In our experiments, we observe that SGD training for these networks does not converge without geographic weighting. We do not apply geographic weighting to DeepNav-distance because it is penalized less for predicting a slightly wrong distance estimate far away from the destination by use of square-root of the distance as the label.

5. Results

In this section, we evaluate the ability of the various DeepNav models to navigate unknown cities and reach the nearest destination and compare them with the algorithm presented in [19]. For reference, we also present the metrics for A* search - note that A* search has access to the entire city graph and destination location while planing the path, while the other methods only have access to images from the agent's current location.

5.1. Baselines

The algorithm for navigating to the nearest McDonald's presented by Khosla et al. in [19] serves as our first baseline. This algorithm extracts Histogram of Oriented Gradient [9] features densely over the entire image and applies K-means to learn a dictionary of size 256. It then uses locality-constrained linear coding [35] to assign the descriptors to the dictionary in a soft manner, and finally builds a 2-level spatial pyramid [21] to obtain a final feature of dimension 5376. We use the publicly available code from the authors [1] to compute the features. To speed up dictionary creation, we create it from a collection of 18,000 images sampled randomly from all the training cities (3000 from each city). A Support Vector Regressor (SVR) [10, 34] is learnt to map an input image to the square root of the distance to the nearest destination establishment in the 90° arc in the direction pointed to by the node. We chose the regularization constant in the SVR by picking the value which minimized the Euclidean error over the 4 test cities (see next section). Our second baseline is a random walk algorithm. This algorithm picks a random action at each node.

5.2. Experimental setup

We train the DeepNav models on 6 cities (Atlanta, Boston, Chicago, Houston, Los Angeles, Philadelphia) and test them on the held-out 4 cities (Dallas, New York, Phoenix, San Francisco). This avoids the bias of training and testing on disjoint parts of the same city, and tests whether the algorithms are able to learn about structure in cities and use that knowledge in unknown environments. For each test city, we sample 10 starting locations uniformly around each destination with an average path length d_s. All our agents start at these locations facing a random direction and navigate the city using the inference procedures described in §4. To prevent looping, the agent is not allowed pick the same action twice from a node. If the agent has no option to move from a location, it is re-spawned at the nearest node with an open option to move. The agent is considered to have reached a destination if it visits a node within 75m of it, and the maximum number of steps is set to 1000.

5.3. Evaluation metrics

We use the two metrics proposed in [19]: 1) success rate (fraction of times the agent reaches its destination) and 2) average number of steps taken to reach the destination in the successful trials. To ease comparison of various methods,

Method	Dallas					New York					Phoenix					San Fancisco					Mean
	B	C	G	H	M	B	C	G	H	M	B	C	G	H	M	B	C	G	H	M	
Random walk	29.24	34.28	47.43	17.73	39.40	58.90	53.93	38.57	45.00	45.37	25.63	36.23	33.73	33.98	29.33	44.47	54.04	45.88	40.00	42.21	39.77
A*	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.0
HOG+SVR [19]	48.48	56.63	62.00	06.06	42.86	76.03	**81.55**	54.29	**77.27**	70.37	31.25	41.56	49.37	44.90	28.89	66.67	**70.32**	**73.40**	57.32	53.49	54.63
DeepNav-distance	27.27	**60.24**	68.00	30.30	45.24	**80.82**	66.99	48.57	68.18	69.63	37.50	54.55	49.37	46.94	35.56	63.83	**70.32**	67.55	**68.29**	**65.12**	56.21
DeepNav-direction	33.33	36.14	40.67	06.06	33.33	62.33	50.49	**57.14**	45.45	58.52	25.00	35.06	56.96	**55.10**	31.11	45.39	61.64	60.64	42.68	13.95	42.55
DeepNav-pair	**54.55**	45.78	**78.67**	**48.48**	**59.52**	**80.82**	67.96	**57.14**	70.45	**73.33**	**43.75**	**55.84**	**64.56**	51.02	**48.89**	**69.50**	66.21	72.87	52.44	37.21	**59.95**

Table 2: Success rates of various algorithms, $d_s = 470m$.

Method	Dallas					New York					Phoenix					San Fancisco					Mean
	B	C	G	H	M	B	C	G	H	M	B	C	G	H	M	B	C	G	H	M	
Random walk	315.11	362.67	299.07	339.28	330.13	309.59	309.04	341.49	380.03	370.48	318.77	349.53	347.89	338.43	302.59	317.05	292.74	319.62	329.60	373.98	332.35
A*	19.12	19.27	16.62	24.03	18.45	16.13	16.72	17.40	17.56	17.35	19.88	21.57	17.97	18.31	24.11	18.09	15.63	17.65	18.80	19.93	18.73
HOG+SVR [19]	244.50	326.53	257.88	164.00	**101.61**	194.09	263.52	198.29	276.24	277.79	349.60	215.22	213.69	258.05	173.85	253.38	262.00	268.59	262.51	249.13	240.52
DeepNav-distance	321.33	289.26	295.07	163.50	303.11	322.95	247.10	220.88	284.03	267.84	270.17	**180.29**	167.69	190.26	162.94	281.57	209.27	279.06	286.59	234.18	248.85
DeepNav-direction	**43.18**	**102.63**	**95.16**	**60.00**	199.50	**113.87**	**55.19**	**132.63**	**127.38**	**156.01**	**132.50**	189.89	**119.71**	**166.44**	**152.71**	**68.55**	**109.44**	**128.99**	**129.29**	**90.17**	**118.66**
DeepNav-pair	414.44	246.03	243.12	433.81	158.32	223.03	239.00	246.65	331.38	269.07	273.14	209.12	174.53	231.64	35.64	222.09	263.86	256.37	226.47	247.38	257.25

Table 3: Average number of steps for successful trials, $d_s = 470m$. Destination abbreviations: B = Bank of America, C = church, G = gas station, H = high school, M = McDonald's.

we propose the expected number of steps metric, which is calculated as $s * L + (1 - s) * L_{max}$ where s is the success rate, L is the average number of steps for successful trials and L_{max} is the maximum number of steps (1000).

The metrics are averaged over all starting locations of a city (and over 20 trials for the random walker). Tables 2 and 3 show the success rate and the average number of steps for successful trials for $d_s = 480m$. We see that DeepNav-pair has the highest average success rate. DeepNav-direction has the lowest average path length for successful trials, but lowest success rate. This indicates that it is effective only for short distances. Table 4 shows the expected number of steps averaged over all destinations and starting locations, for $d_s = 470m$, $690m$ and $970m$. Detailed metrics for $d_s = 690m$ and $970m$ are presented in the supplementary material. DeepNav-pair outperforms the baseline as well as other DeepNav architectures for most starting distances. We hypothesize that DeepNav-pair can perform better because it is the only algorithm that is trained by 'looking' in all directions before choosing. We also note that DeepNav-distance outperforms the agent from [19], indicating the better quality of deep features.

If a model learns visual features common to paths leading to destinations, it should pick the correct direction with high confidence near the destination and at major intersections. For a given location, a measure of the confidence of the model for picking one direction is the variance of scores predicted for all directions. We plot this variance at all locations in San Francisco computed from models trained for navigating to Bank of America in Figure 5. The figure shows empirically that the DeepNav-pair agent chooses one direction more confidently as it nears a destination, while other models show less of this behavior. Another approach to get an insight into the visual features learnt by the algorithms is to see the which images are most (and least) confidently predicted as pointing to the path to destination. In Figure 6, we plot the top- and bottom-5 images (sorted by score) while the DeepNav-pair agent is navigating New York for McDonald's and Dallas for gas station. It can be seen that the CNN correctly learns that center-city commercial areas have a high probability of having a McDonald's establishment, and gas stations are found around intersections of big streets with that have parked cars.

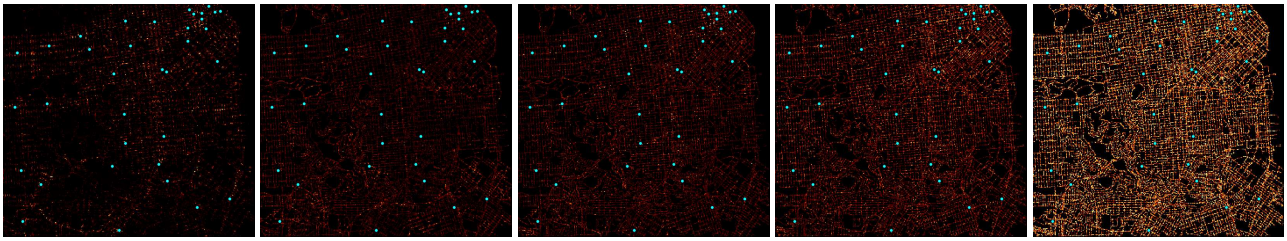
Method	Expected number of steps		
	d_s=470m	d_s=690m	d_s=970m
Random walk	733.99	854.8913	911.85
A*	18.73	27.3204	39.57
HOG+SVR [19]	588.66	705.31	791.93
DeepNav-distance	580.69	**684.22**	773.02
DeepNav-direction	626.28	697.26	780.53
DeepNav-pair	**553.39**	689.04	**766.32**

Table 4: Expected number of steps for various algorithms.

Figure 7 shows some example navigation paths generated by the DeepNav and baseline models while navigating for the nearest church in New York.

6. Conclusion

We presented 3 convolutional neural network architectures (DeepNav-distance, -direction and -pair) for learning to navigate in large scale real-world environments. These algorithms were trained and evaluated on a dataset of 1 million street-view images collected from 10 large cities. We show how A* search can be used to efficiently generate training labels for images for various DeepNav architectures. We find that data-driven deep convolutional features (DeepNav-distance) outperform a combination of hand-crafted features and SVR [19]. In addition, training the network to 'look' in all directions using a Siamese architecture (DeepNav-pair) outperforms networks that are trained to estimate distance to destination (DeepNav-distance) or optimal action (DeepNav-direction).



(a) DeepNav-pair (b) DeepNav-direction (c) DeepNav-distance (d) HOG+SVR [19] (e) Random walk

Figure 5: Confidence of predictions while navigating for Bank of America (blue dots) in San Francisco (a test city). Brighter colors imply higher variance. Concentration of high-variance regions indicates that DeepNav-pair confidence increases near destinations and it effectively learns visual features common to optimal paths.

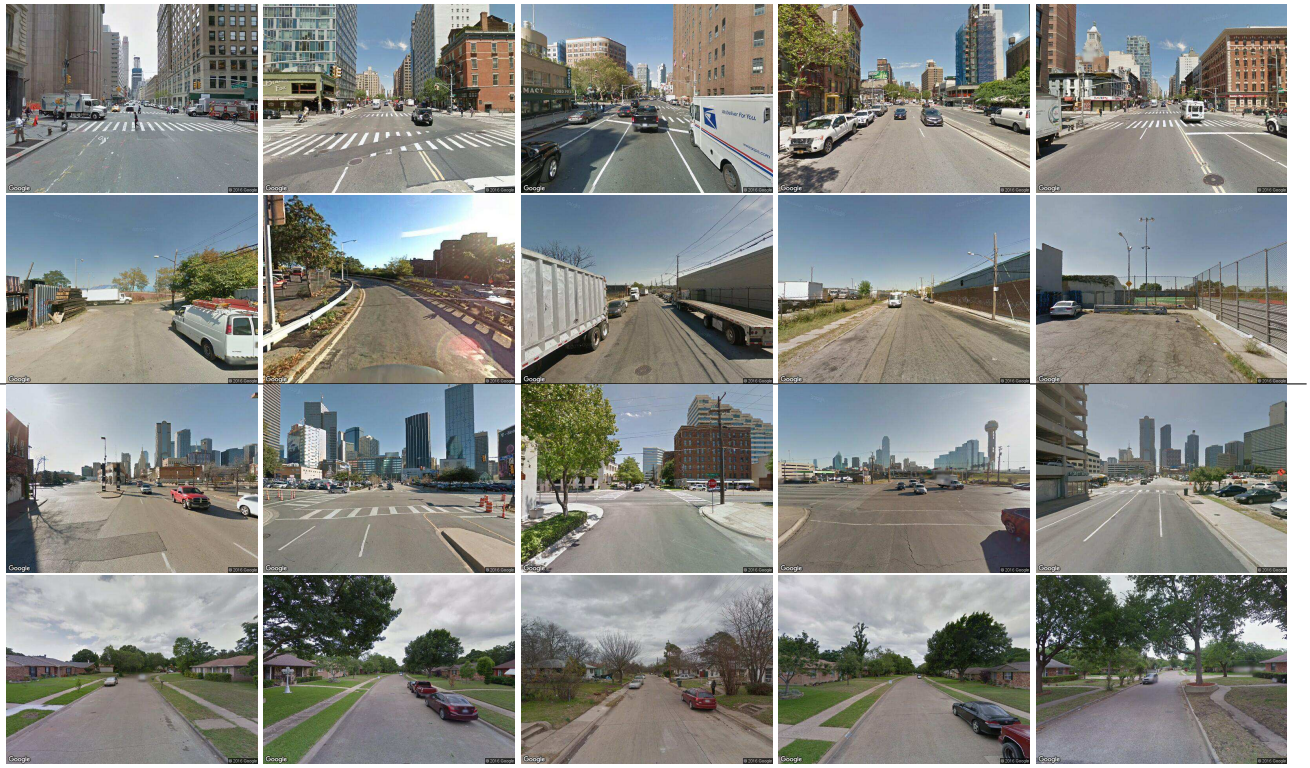
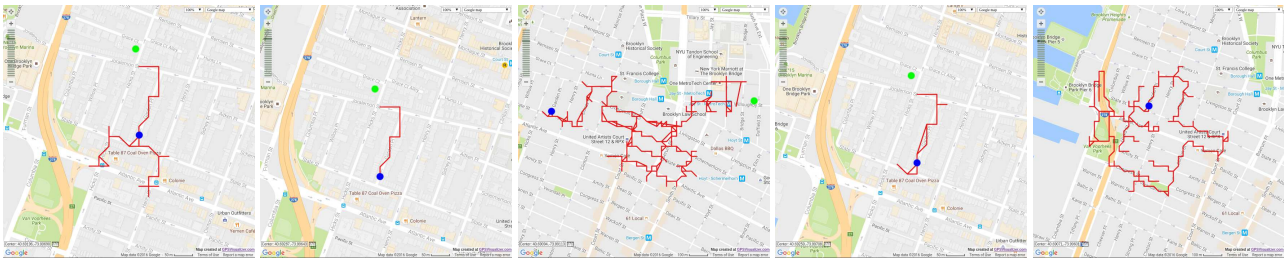


Figure 6: Rows 1-2: Top 5 high scoring and low scoring (respectively) images predicted by DeepNav-pair navigating to McDonald's in New York (a test city). Rows 3-4: Similar images for navigating to gas station in Dallas (a test city).



Found, length = 60
DeepNav-pair

Found, length = 10
DeepNav-direction

Found, length = 520
DeepNav-distance

Found, length = 35
HOG+SVR [19]

Not found
Random walk

Figure 7: Paths for navigating to church in New York (a test city). Blue dot = start, green dot = destination (when found).

References

[1] feature-extraction. https://github.com/adikhosla/feature-extraction, 2016. Accessed: 2016-11-11. 6

[2] Google maps roads API. https://developers.google.com/maps/documentation/roads/nearest, 2016. Accessed: 2016-11-11. 2, 4

[3] Google places API. https://developers.google.com/places/web-service/search, 2016. Accessed: 2016-11-11. 2, 4

[4] A. Aydemir, A. Pronobis, M. Göbelbecker, and P. Jensfelt. Active visual object search in unknown environments using uncertain semantics. *IEEE Transactions on Robotics*, 29(4):986–1002, 2013. 3

[5] R. Bajcsy. Active perception. *Proceedings of the IEEE*, 76(8):966–1005, 1988. 3

[6] G. J. Brostow, J. Shotton, J. Fauqueur, and R. Cipolla. Segmentation and recognition using structure from motion point clouds. In *European conference on computer vision*, pages 44–57. Springer, 2008. 1

[7] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille. Semantic image segmentation with deep convolutional nets and fully connected crfs. *arXiv preprint arXiv:1412.7062*, 2014. 2

[8] W. Choi, Y.-W. Chao, C. Pantofaru, and S. Savarese. Understanding indoor scenes using 3d geometric phrases. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 33–40, 2013. 1

[9] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, volume 1, pages 886–893. IEEE, 2005. 2, 6

[10] R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin. Liblinear: A library for large linear classification. *Journal of machine learning research*, 9(Aug):1871–1874, 2008. 6

[11] R. Girshick. Fast R-CNN. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1440–1448, 2015. 2

[12] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In *JMLR W&CP: Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics (AISTATS 2010)*, volume 9, pages 249–256, May 2010. 6

[13] R. Hadsell, P. Sermanet, J. Ben, A. Erkan, M. Scoffier, K. Kavukcuoglu, U. Muller, and Y. LeCun. Learning long-range vision for autonomous off-road driving. *Journal of Field Robotics*, 26(2):120–144, 2009. 1

[14] J. Hays and A. A. Efros. Im2gps: estimating geographic information from a single image. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pages 1–8. IEEE, 2008. 2

[15] D. Hoiem, A. A. Efros, and M. Hebert. Geometric context from a single image. In *Tenth IEEE International Conference on Computer Vision (ICCV'05) Volume 1*, volume 1, pages 654–661. IEEE, 2005. 2

[16] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*, 2014. 6

[17] D. Joho, M. Senk, and W. Burgard. Learning search heuristics for finding objects in structured environments. *Robotics and Autonomous Systems*, 59(5):319–328, May 2011. 3

[18] A. Kendall, M. Grimes, and R. Cipolla. Posenet: A convolutional network for real-time 6-dof camera relocalization. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2938–2946, 2015. 2

[19] A. Khosla, B. An, J. J. Lim, and A. Torralba. Looking Beyond the Visible Scene. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Ohio, USA, June 2014. 1, 2, 4, 6, 7, 8

[20] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012. 2

[21] S. Lazebnik, C. Schmid, and J. Ponce. Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories. In *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, volume 2, pages 2169–2178. IEEE, 2006. 6

[22] G. Li and Y. Yu. Visual saliency based on multiscale deep features. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5455–5463, 2015. 2

[23] Y. Li, M. Paluri, J. M. Rehg, and P. Dollár. Unsupervised learning of edges. In *CVPR*, 2016. 2

[24] Z. Liu, P. Luo, X. Wang, and X. Tang. Deep learning face attributes in the wild. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 3730–3738, 2015. 2

[25] J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3431–3440, 2015. 2

[26] P. Mirowski, R. Pascanu, F. Viola, H. Soyer, A. Ballard, A. Banino, M. Denil, R. Goroshin, L. Sifre, K. Kavukcuoglu, et al. Learning to navigate in complex environments. *arXiv preprint arXiv:1611.03673*, 2016. 3

[27] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu. Asynchronous methods for deep reinforcement learning. *arXiv preprint arXiv:1602.01783*, 2016. 2, 3

[28] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013. 2

[29] R. Mottaghi, X. Chen, X. Liu, N.-G. Cho, S.-W. Lee, S. Fidler, R. Urtasun, and A. Yuille. The role of context for object detection and semantic segmentation in the wild. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 891–898, 2014. 1

[30] J. Oh, V. Chockalingam, S. Singh, and H. Lee. Control of memory, active perception, and action in minecraft. *arXiv preprint arXiv:1605.09128*, 2016. 2, 3

[31] S. Ren, K. He, R. Girshick, and J. Sun. Faster R-CNN: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015. 2

[32] S. Shankar, V. K. Garg, and R. Cipolla. Deep-carving: Dis-

covering visual attributes by carving deep neural nets. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3403–3412, 2015. 2

[33] K. Simonyan and A. Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. *CoRR*, abs/1409.1556, 2014. 3, 4, 6

[34] A. Smola and V. Vapnik. Support vector regression machines. 1997. 2, 6

[35] J. Wang, J. Yang, K. Yu, F. Lv, T. Huang, and Y. Gong. Locality-constrained linear coding for image classification. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 3360–3367. IEEE, 2010. 6

[36] R. F. Wang and E. S. Spelke. Human spatial representation: Insights from animals. *Trends in cognitive sciences*, 6(9):376–382, 2002. 5

[37] T. Weyand, I. Kostrikov, and J. Philbin. Planet - photo geolocation with convolutional neural networks. In *European Conference on Computer Vision (ECCV)*, 2016. 2

[38] S. Zheng, S. Jayasumana, B. Romera-Paredes, V. Vineet, Z. Su, D. Du, C. Huang, and P. Torr. Conditional random fields as recurrent neural networks. In *International Conference on Computer Vision (ICCV)*, 2015. 1, 2

[39] B. Zhou, A. Lapedriza, J. Xiao, A. Torralba, and A. Oliva. Learning deep features for scene recognition using places database. In *Advances in neural information processing systems*, pages 487–495, 2014. 2

[40] Y. Zhu, R. Mottaghi, E. Kolve, J. J. Lim, A. Gupta, L. Fei-Fei, and A. Farhadi. Target-driven visual navigation in indoor scenes using deep reinforcement learning. *arXiv preprint arXiv:1609.05143*, 2016. 2