Predicting Absolute Size and Depth of K-Objects in a Still Monocular Image using Deep Neural Nets

Rajat Bhageria RAJAT@SEAS.UPENN.EDU
Kashish Gupta KASHISHG@WHARTON.UPENN.EDU
Li He LIHE@SEAS.UPENN.EDU

1. Introduction

Absolute size (height and width) and depth estimation of objects in an image has important applications in computer vision (e.g. fitting clothing of the right size based on a simple image). In fact, given the focal length of the camera that took the image and the depth of the objects in the image, finding the size of the objects is a simple linear algebra problem. However, in this project, our goal is to learn focal length using a training set of 1449 RGBD image so that an algorithm could, for instance, predict the correct size of a dress size without knowing the focal length of the lens that took the picture.

We will estimate the absolute size of k objects in each image using both linear regression and deep neural net (DNN) methods by learning the focal length and predicting the absolute height and width of the object in terms of meters, the results of which will be compared. A second question that we seek to tackle is to find the absolute depth given only an RGB image in the first place; to solve this we designed a convolutional neural net (CNN) to learn depth perception given only RGB images.

2. Background

2.1. Depth Perception Background

Up to this point, most of the work in depth perception has been in one of three general categories (Saxena & Ng, 2008): 1) Binocular depth using two images or multiple cameras. This is how our eyes perceive depth; 2) Depth using optical flow or a video where once again we have access to multiple frames; 3) Depth from de-focusing.

Using binocular vision and depth detection is relatively simple. Some tools include stereopsis and convergence of the two images / eyes at a single object. There are various monocular depth tools such as texture, parallax, and relative sizing that humans use as well. However, the problem with monocular cues is that they provide relative information, preventing the inference of absolute values of particular objects. One group at Stanford (Saxena & Ng, 2006) led by Andrew Ng has managed to predict absolute depth in meters using random Markov chains. We are seeking to use CNNs to solve this problem.

2.2. Absolute Size Estimation Background

Estimating an objects absolute size remains a common computer vision problem using homogeneous coordinates. This can be computed using focal length and knowledge of the depths of the objects. In other words, given the pixel

$$\lambda \begin{bmatrix} u_{\text{img}} \\ v_{\text{img}} \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & & p_x \\ & f_y & p_y \\ & & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}$$

Equation 1: Finding the X and Y locations in space of a pixel $\{u_{img}, v_{img}\}$ given the depth Z of the pixel in meters, and the focal length of the camera in pixels.

location of an object in an image, the focal length of the camera, the pixel location of the center of the camera, and the depth of the image Z in meters, we can estimate the actual location of the image in meters. Once the focal length of the camera used to take the image is known, finding the actual height and width of an object in meters is a simple, well known task.

However, in this project, the goal is to estimate the size by abstracting out the focal length of the images.

3. Methodology

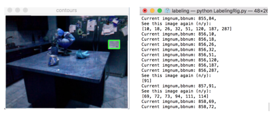
3.1. Estimate Size of K-Objects given an RGBD image

This part answers the question, given you have an RGBD image, can you find the sizes of each of the objects in the image?

1. Draw bounding boxes around all the objects in each of the 1449 training images. Then, and then find the average absolute depth of the each of the objects in the image. Use OpenCV to generate bounding boxes around all the training images and test images. This is using

a Fast-RCNN model to find the bounding boxes (Santos, 2017). Keep track of the heights and the widths of the bounding boxes.

2. Using OpenCV, generate labels for 200 training images to train the size estimation algorithms. The dataset of images (Silberman, 2012) contains many common office and household objects. Thus, by selecting the bounding box around objects of interest, we are able to manually assign height and width. It's important to build an easy-to-use command line interface to reduce the labeling time down to around three images per minute.



```
Current imgnum,bbnum: 855,84,
See this image again (n/y):
[10, 18, 26, 32, 51, 120, 187, 287]
Current imgnum,bbnum: 856,10,
Current imgnum,bbnum: 856,18,
Current imgnum,bbnum: 856,26,
Current imgnum,bbnum: 856,32,
Current imgnum,bbnum: 856,51,
Current imgnum,bbnum: 856,120,
Current imgnum,bbnum: 856,187,
Current imgnum,bbnum: 856,287,
See this image again (n/y):
[91]
Current imgnum,bbnum: 857,91,
See this image again (n/y):
[69, 72, 73, 94, 111, 114]
Current imgnum,bbnum: 858,69,
Current imgnum,bbnum: 858,72,
```

Figure 1. Methodology for labeling images

3. Do one of two approaches to find the absolute sizes of the k objects in each of the n test images: 1) use linear regression and 2) use deep neural nets. The goal is to predict which of these ML algorithms has a better predictive accuracy.

4. **Approach 1** to find size in meters of the k objects: Use linear regression to learn the focal length of the camera in order to estimate the size of the objects in terms of height and width. In this geometric approach, if we know the depth (Z direction) in meters of an object, and we know the pixel height and width (which we do because were drawing bounding boxes around the image), we can figure out the height and width of the object in meters using the expressions: However, since were given arbitrary images where we

$$u_{img} = u_{ccd}\frac{w_{img}}{w_{ccd}} + p_x = \underbrace{f_x}_{\text{Focal length in pixel}}\frac{X}{Z} + p_x$$

$$v_{img} = v_{ccd}\frac{h_{img}}{h_{ccd}} + p_y = \underbrace{f_y}_{\text{Focal length in pixel}}\frac{Y}{Z} + p_y$$

Equation 2: *The non vectorized form or Equation 1 used to compute the actual heights and widths given the pixel heights and widths.*

dont know the focal length (or the height / width of the camera sensor), we cannot take a purely geometric approach. However, since this is a linear transformation, we can simply use linear regression in order to learn the focal length so that we can predict the size of the objects in meters.

Training Data: The n=80 size labeled RGBD images generated from step 3

Test Data: An additional n=6 RGBD images from the NYU dataset in which we know the pixel heights and widths of the k objects in each image and the depth of each of the objects but dont know the sizes of the objects.

Train two models using linear regression to find the width and height of the objects:

Model 1 (for the width of objects):

$$X = \frac{1}{f}(u_{img} - p_x) * Z$$

where u_{img} is the pixel width of object k and p_x is the width of the image in pixels /2.

Input Features: u_{img} (width of the bounding box in pixels; p_x (the x coordinate of the center of the image); Z (the depth of the object in the bbox in meters); Y_{train} (width in meters of object)

Outputs Classes: u_{img} (width of the test bounding box in pixels); p_x (the x coordinate of the center of the test image); Z (the depth of the test object in the bbox in meters); Y_{test} expected (width in meters of new object)

Model 2: exactly the same as Model 1 except for the height of objects and thus X_{train} is v_{img}, p_y, and Z.

5. **Approach 2** to find size in meters of the k objects: Train a Deep Neural Net (DNN) Regressor that takes in the absolute depth of the k objects / people in the image and return the absolute height and width of the object / person.

Input Features: X_{train}: u_{img} (height of the bounding box in pixels); v_{img} (width of the bounding box in pixels; p_x (the x coordinate of the center of the image); p_y (the y coordinate of the center of the image); Z (the depth of the object in the bbox in meters). Y_{train}: width and height in meters of object

Outputs Classes: The absolute width and height of the object in meters

Training Data: The n=80 size labeled RGBD images generated from step 3

Test Data: An additional n=6 RGBD images from the NYU dataset in which we know the pixel heights and widths of the k objects in each image and the depth of each of the objects but dont know the sizes of the objects.

3.2. Estimate the Depth of an image given RGB image

This part asks the question, given you have a RGB image find the depth in meters of each of the pixels in the image.

Solution: Train a CNN that takes in an image and returns the absolute depth in meters of each pixel in the image.

Training Data: This will be trained using the NYU datasets providing RGB-D data (raw image to absolute depth per pixel data) for 1449 images that was taken using the Microsoft Kinect Camera. This dataset has NxMx3 images and contains NxM labeled depth maps in meters.

Test Data: we can use the portion of the dataset as test data because the grount truth depth for all pixels in the image will already be known, which will provide an easy method for calculating our prediction error.

Input Features: A NxMx3 RGB images

Output: A NxM depth image with depth in meters for each of the pixels in the image

Methodology: Using Tensorflow, create a convolutional neural net and vary the number of convolutional layers followed by RELU layers to introduce nonlinearities into the images, max pooling layers, and fully connected layers to classify depth between 0-10 meters. Calculate loss at each iteration and back-propagate to train the net at each iteration, until the losses converge.

Implementation: Taking in a 480x640x3 image in batches, the first step was to run a convolution layer with a 5x5 kernel creating 2 filters with the relu activation. Then every 4th pixel was subsampled from the image, and the process was repeated with 4 filters and every 2nd pixel. The 2D matrix was then flattened into a linear vector and two fully connected layers were used as the final layers, with a dropout of 0.4 in between. The output was to predict depth for each of the 307,200 pixels in the image, and then to calculate loss using L2 loss, essentially the L2 norm between our prediction and the ground truth. The optimizer minimizes loss until the model converges or runs for 1000 iterations.

4. Discussion and Problems Encountered

4.1. Bounding Boxes for Size Estimation

The bounding boxes drawn assumed that the object is situated on a flat plane perpendicular to device capturing the photo. As a result, even when the dimension of the objects is known, there are instances where the bounding box captures the object that is on a very large slant, and would not hold for the equation used in the geometric approach. The computer vision computation for finding bounding boxes relies on edge detection. Many unimportant bounding boxes were detected, so we set thresholds to remove them.

4.2. Linear Regression for Size Estimation

As we labeled more training data, we found mistakes in our linear regressor:

Initially, our training data mainly consisted of A4 pieces of paper across around 20 different images. As a result of this, our linear regression model learned a strong bias towards A4 sized objects. The linear regressor was predicting to the size of a paper. The lack of variability in the training instances meant that the model was not generalize to different sized objects.

Initially, we kept u_img, p_x, and depth as three separate feature input into the model. We thought that separating the feature will provide more data points for the model to train on. However, we were quick to realize that the linear regression algorithms only reconcile the translational relationships and not the multiplicative ones. Thus, our regressor worked a lot better when we combined the input into one feature, and trained against that.

Using the closed form equation vs. gradient descent. We originally used the gradient descent form of linear regression since we had very little data, but with larger amounts of data realized a closed form equation would yield similar results with considerable efficiency gains.

4.3. Neural Nets for Size Estimation

Initially we were feeding in the height in pixels, the width in pixels, and depth and were asking to predict both the height and the width in meters. However, we found that because we only had around 100 labeled data points, the net was putting more weight on either the height or the width. As a result, we decided to train two different nets, one that predicts only height and one that only predicts width.

Even after creating two separate neural nets to predict height and width, the pixel center coordinates of the image (P_x and P_y) were overemphasized in our model because they were the exact same for every image. We modified the equations to remove these features.

Now we pass in pixel height and width of the objects into the neural net. However, since we once again had a small amount of training data and found that the algorithm was emphasizing either the depth or the width rather than learning the right correlation between them (i.e., uimg = focal length * width/depth). Similarly to linear regression, we now had to convert the two inputs to a single input by setting height as (pixels)/depth (meters).

We had difficulty in creating a layers for a deep neural net even though we had one input (or in earlier iterations 5, 3, 2 features) and only one output. In many iterations we tried having five-six layers of one neuron each and we saw that the values for each of the test inputs converged and predicted the same result. In reality, a deep neural net should have been able to solve each of these issues but the simple fact that we only had n=80 training instances made the training for the DNN quite hard and also inaccurate.

4.4. CNN for Depth Estimation

In our readings, namely the paper Learning Depth from Monocular Images by Saxena, Chung, and Ng, we read about several methods for learning depth using traditional approaches to machine learning such as Markov Random Fields. We wanted to try to achieve the same results using a convolutional neural net, but the memory constraints on Biglab and our personal computers became prohibitive. The tensors would train only for one iteration before running into memory errors, even after decreasing the filters from 64 to 4 and subsampling every 4th pixel instead of every 2nd. To fix this, we would use smaller images next time, or we would try to classify certain quadrants of the image for their depth rather than every single pixel.

In addition, configuring our convolutional neural net to predict depth was many parts guesswork and many parts testing. Many of the existing classifiers that we took inspiration from were for object recognition, whereas the layers used in our model had to be different. Given more time, we would have tried many more permutations of layers with different activation functions and loss calculations.

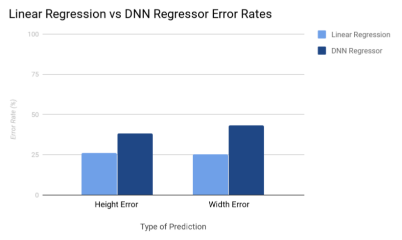
Ultimately, we were able to conduct the rest of our experiment with the dataset we already had because depth values were computed as ground truth. In addition, extracting depth from monocular images has already been accomplished in several papers we we did not focus on this for our project, as we provide a novel approach to size detection.

5. Evaluation of Results

5.1. Linear Regression vs DNN to predict size of objects in an RGBD image

Linreg errors: 26.2% error for Height prediction; 25.1% error for Width prediction

Neural errors: 38.2% error for Height prediction; 43.4% error for Width prediction Interestingly, we found that the



simple linear regression algorithm had a higher accuracy than a DNN. The simple reason for this is that while the linear regression algorithm only needs two points to fit a line, a DNN needs many orders more. Since we only had n=80 instances of training data, it seems reasonable that the DNN wasnt as accurate as linear regression.

5.2. CNN to predict depth of pixels given RGB image

Though we were only able to train the CNN for a couple iterations at a time, the $log_1 0$ of the loss at these iterations was 6, which is very low for a loss calculating the sum squared distances between all the 307,200 predictions and labels. If allowed to run for up to 1000 iterations, and also with a larger number of nodes in the fully connected layers, we could expect the weights in this CNN to converge and for predictions to be meaningful. We were not able to test with multiple iterations because of memory constraints.

6. Future Work

Improve bounding boxes: E.g., integrating depth and nearby pixels to estimate bounding boxes. For objects that are not situated on a perpendicular plane, we integrate in the difference in depth (by constructing a pythagorean triangle to reconcile the slant). This would better suit our input data. The bounding box computer vision computation does not put bounding boxes around some objects that we could classify.

Have more test images: Because of the large amount of time it takes to size label objects in images, we tried to maximize the number of labeled images in our training set but this meant that we only had n=6 images in test which skewed our accuracy.

Standardization of inputs: one major problem we realized after the fact was that when we were passing in Px,Py (the center of the image) as a feature into our linreg/DNNRegressors, they were about three orders of magnitude larger than the depths (in meters). And thus, our models were skewed and we had to abstract out Px,Py and not feed them in as a feature. Instead, we could have standardized these features.

Train on multiple focal lengths: since we only had access to the NYU dataset as a training dataset, we had to settle for only having images with a couple different focal lengths. In the future, we would like to have images taken from cameras with different focal lengths to prevent our algorithms from overfitting so much to the provided training set.

Have more computing power: One of the major problems while using CNNs to predict depth was physical memory on the machine because a densely connected layer had 307,200 outputs for even a small image.

Acknowledgments

Tutorials Followed include:

https://www.tensorflow.org/tutorials/layers

https://gist.github.com/bigsnarfdude/
d811e31ee17495f82f10db12651ae82d

https://www.tensorflow.org/versions/master/get_started
/linear_regression

https://leonardoaraujosantos.gitbooks.io/artificial-
inteligence/content/object_localization_and_detection.html

http://adventuresinmachinelearning.com/convolutional
-neural-networks-tutorial-tensorflow/

https://github.com/aymericdamien/TensorFlow-
Examples/blob/master/examples/3
_NeuralNetworks/convolutional_network.py

https://github.com/aymericdamien/TensorFlow-
Examples/blob/master/examples/3_NeuralNetworks/
convolutional_network.py

https://www.tensorflow.org/get_started/mnist/pros

https://www.tensorflow.org/get_started/input_fn

https://www.tensorflow.org/tutorials/linear#feature
_columns_and_transformations

https://www.tensorflow.org/api_docs/python/
tf/contrib/learn/DNNEstimator

https://www.tensorflow.org/versions/master/api
_docs/python/tf/estimator/Estimator

https://stackoverflow.com/questions/42900824/ how-to-change-hidden-units-in-dnnregressor- in-the-example-posted-on-tensorflow-e

https://johnflux.com/2017/02/20/practical-deep-neural-network- in-a-few-lines-of-code-with-tensorflow/

References

Santos, Leonardo Araujo. Object localization and detection. *Object Localization and Detection Artificial Inteligence*, 2017.

Saxena, Ashutosh, Sung H. Chung and Ng, Andrew Y. Learning depth from single monocular images. *Advances in neural information processing systems*, 2006.

Saxena, Ashutosh, Sung H. Chung and Ng, Andrew Y. 3-d depth reconstruction from a single still image. *International Journal of Computer Vision*, 76(1):53–69, 2008.

Silberman, Nathan, et al. Indoor segmentation and support inference from rgbd images. *Computer VisionECCV*, pp. 746–760, 2012.

Absolute Size and Depth Detection in Monocular Images

Inspiration: Given a regular still image, people are unable to tell the actual size of objects without depth of the object and focal length of the camera. Machine learning may be able to fit both of these variables.

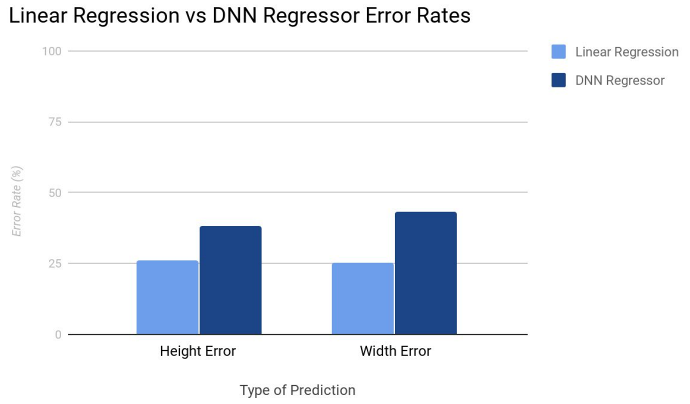
Impact: Augmented reality applications could provide virtual fitting rooms to place real sized objects on subjects. The height and width of a subject could be predicted given depth.

Previous Methods: Reconstructing a depth map from a still image using traditional approaches such as a Markov Random Model with a Gaussian and Laplacian model

Novelty: Convolutional neural nets have not been used to classify depth per pixel for an image. We then use 2 methods to predict absolute size based on depth data: a linear regressor and a deep neural net.

Dataset: RGB-D data for 1449 images provided by NYU

Results: Using Linear Regression: 26.2% error for Height prediction; 25.1% error for Width prediction. Using Deep Neural Nets: 38.2% error for Height prediction; 43.4% error for Width prediction



Rajat Bhageria, Li He, Kashish Gupta

Absolute Size and Depth Detection in Monocular Images

Method: CNN For Estimating Depth: Using RGBD dataset train a CNN using Tensorflow where inputs are images and outputs are depth maps for image.

Method: Linear Regression to Estimate Size of Objects: Create bounding boxes around objects in image, manually label RGBD dataset images with object sizes, and then use closed form linear regression with pixel height/pixel width and depth of object in meters to learn focal length and predict height/width of object.

Method: Deep Neural Net Regressor to Estimate Size of Objects: Exact same method as using linear regression however use a DNN Regressor with two hidden layers to predict.

Problems: In terms of size estimation, since there were only 80 images to train with, the linear regressor actually had a higher test accuracy than the DNN. In terms of depth estimation using CNNs, because of memory limitations we were only able to run one image and had a cost of 6.

Future Work: Have more training data, predict on more test images, have greater variability in objects detected in the images, standardize inputs, and train over a greater variety of focal lengths (cameras from which training images are taken).

$$u_{img} = u_{ccd}\frac{w_{img}}{w_{ccd}} + p_x = \underbrace{f_x}_{\text{Focal length in pixel}}\frac{X}{Z} + p_x$$

$$v_{img} = v_{ccd}\frac{h_{img}}{h_{ccd}} + p_y = \underbrace{f_y}_{\text{Focal length in pixel}}\frac{Y}{Z} + p_y$$

Rajat Bhageria, Li He, Kashish Gupta