CS 543 - Progress Report

Dario Aranguiz
Cu-Khoi-Nguyen Mac

May 11, 2016

1 Updated statement of the project definition and goals

We have no particular changes to our project scope. Our project proposal is copied below for posterity.

For our final project, we plan on implementing and exploring extensions of the 2015 paper *Multimodal Deep Learning for Robust RGB-D Object Recognition* [1] by Eitel et al. Our research group has recently taken interest in the capabilities of depth cameras, and as such we have various time of flight and structured-light depth sensors. Our group members have done quite a bit of work in facial recognition and related areas using depth, but an area we have yet to explore is augmenting traditional RGB-based object recognition approaches with depth information. To this end, we have chosen a recent paper from IROS 2015 that approaches the age-old problem of object recognition with two new modalities, depth and deep learning.

The motivation for this technique is as follows. Convolutional Neural Networks (CNNs) have proven remarkably useful for object recognition in the last few years, with large companies such as Microsoft and Google pouring resources into training more efficient networks. We are also interested in this problem, but unlike Microsoft and Google, we do not have a GPU farm at our disposal to train networks ad-infinitum. This prompts an interesting question; how can we utilize pre-trained ImageNet networks while incorporating depth information? CNNs require a fixed-size input, and most pre-trained networks have only been trained on three-channel information. That constraint is the primary motivation behind Eitel et al's technique.

To get around this problem, two separate three-channel networks are used: one to process the incoming RGB images, and another to process the incoming depth information. This depth information is transformed from single-channel distance information to a three-channel RGB image using a depth-to-color heatmap, the intuition being that useful features such as edges and corners are preserved in this transformation. The final fully-connected layers of each of these networks are then removed, and the second-to-last activation layers are fed into a single large classification layer, merging the two networks. Additional preprocessing is also done, but the system functions loosely as described.

Our expected final outcome is to have a fully-functioning system that can distinguish between objects from the Washington RGB-D Object Dataset. At minimum, we aim to replicate the paper and get a recognition system of our own running locally. We have a few stretch goals, listed in order of expected difficulty:

1. **Different Backends -** The current most-frequently used network backend is a package called Theano. However, Google recently released their own open-source Machine Learning backend called TensorFlow. It would be interesting to compare the efficacy of these two systems with our project.

2. **Different Classification Layers -** In the paper, the authors recommend using a single fully-connected layer at the end of the pipeline for classification. It would be interesting to see how this scheme compares with other classification methods such as logistic regression, SVM, Random Forests, and others.

3. **Different Preprocessing Schemes -** The authors prescribe a number of preprocessing techniques to convert depth images from one-channel data to three-channel data, but other preprocessing techniques have been used in the past to do this as well.

4. **Network Modifications -** This is by and large our most ambitious stretch goal. The two-network system is core to the authors' architecture, but we would be interesting in exploring different ways of fusing depth data with traditional RGB data. This comes with a number of challenges, the least

being that any modification to a pretrained ImageNet network would eliminate the benefit of using a pretrained network at all.

2 Current member roles and collaboration strategies

As mentioned in the proposal, this project is not easily split into modules. Dario has been helping with the actual Keras implementation of the stream network, and Mac has been completing the preprocessing stages as well as the actual Keras implementation. Code is shared in a GitHub repository, whereas copies of the dataset are only stored locally. As for communication, we maintain an open communication channel using Slack as well as simply sitting near each other in our group's office. To that end, we do not have regular meetings, but rather ad-hoc meetings whenever one of us gets stuck.

3 Proposed approach

As mentioned during the initial project proposal, we intend to implement the approach detailed by [1]. A loose description our pipeline is as follows:

1. First, preprocess the images. This involves converting the single-channel depth images into three-channel RGB images so they can be processed by preexisting trained networks such as AlexNet or VGG. Before colorization, these images are segmented using a human-drawn mask provided in the Washington dataset and padded into a 227 × 227 square. More details and sample output from our system in Section 5.1.
2. Next, train the network. This is done in multiple stages:
   (a) First, we consider two separate networks with the same model architecture; one corresponding to the color image and one corresponding to the colorized depth image. These networks are initialized with AlexNet pretrained model weights and trained individually using a standard stochastic gradient descent method provided by Keras. An example of this training method can be found on the Keras Examples page [2]. AMore details on the model architecture in Section 5.2.
   (b) Next, we strip the final `softmax` classification layer from each of the two networks, leaving a 4096-dimension fully-connected layer at the end of each network. These are then fused together as in Figure 5 and trained as a single network using initial weights from the prior step. Keras provides a simple API to concatenate multiple models.
3. Finally, testing is performed using Keras's `model.test()` API.

4 Data

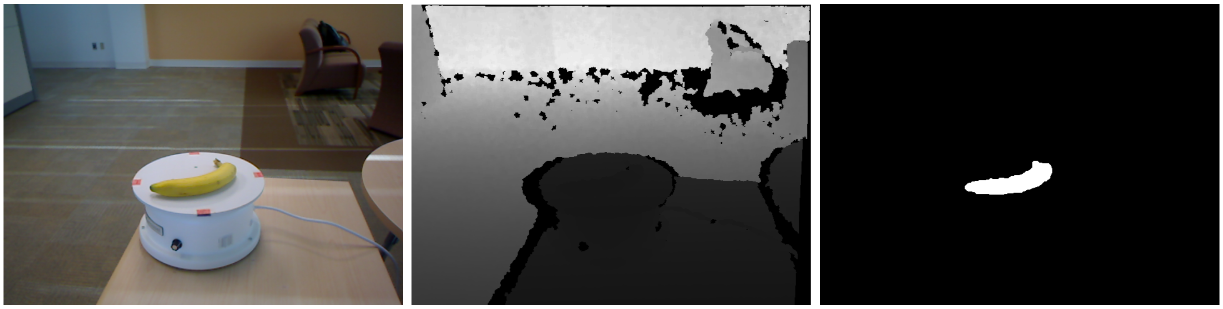
Following the proposed approach by Eitel et al. [1], we decide to use Washington's RGB-D object dataset [3] for training and testing. The dataset contains 300 household objects (instances) divided into 51 categories. Each instance has three different takes, each records a full turn- around rotation of the object. A sample of the full database includes images of color, depth, mask (as seen in Figure 1), and a text file recording the top-left corner of the object.

To speed up the process of testing our system's runnability, we pick out 15 instances from 5 different categories (3 each), namely `apple`, `ball`, `banana`, `bell_pepper`, and `binder`. After the system is fully functional, we will expand the training list to the whole dataset.

5 Initial results

5.1 Data preprocessing

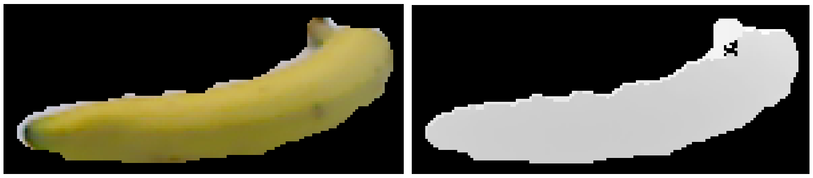
We use the provided masks and top-left corners to remove unnecessary information from color and depth images (Figure 2). Top bottom-right corner if dynamically found, depending on each sample. According



(a) Color image (b) Depth image (c) Mask

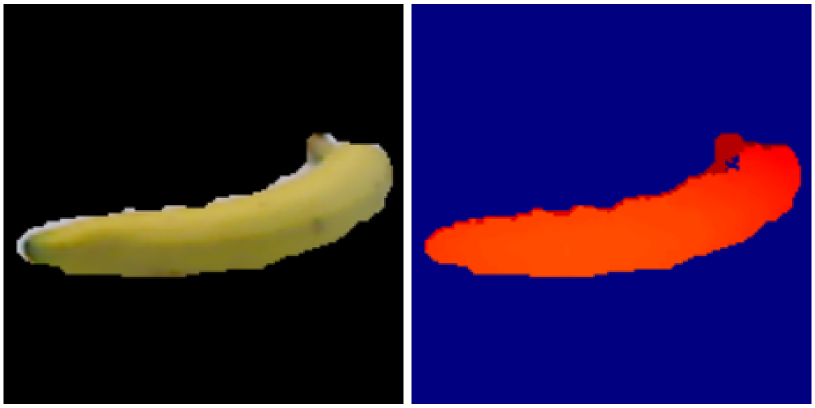
Figure 1: A banana sample of Washington's RGB-D object dataset. The depth image is rescaled to the range [0, 255] to visual the information.

to Eitel et al. [1], they convert the depth images into color ones using colorjet mapping as they want to have the same training mechanism for both color and depth. We inherit this idea and apply the same depth colorizing technique. The images are then rescaled to 227 × 227 (Figure 3) by replicating the longer side from the cropped images. This help to maintain the object in the center without deformation after rescaling.



(a) Color image (b) Depth image

Figure 2: Cropping banana sample after applying mask.



(a) Color image (b) Depth image (after colorizing)

Figure 3: Rescaling banana sample to 227x227 by replicating the longer side.

5.2 Model architecture

Figure 4 illustrates the network architecture proposed by Eitel et al. [1]. There are two separate stream models, corresponding to color and depth channels. Each stream contains 5 convolution (from `conv-1` to `conv-5`) and 2 fully connected layers (`fc6` and `fc7`). The streams are fused together by one fusion layer (`fc-fus`) and classified by `class` layer.

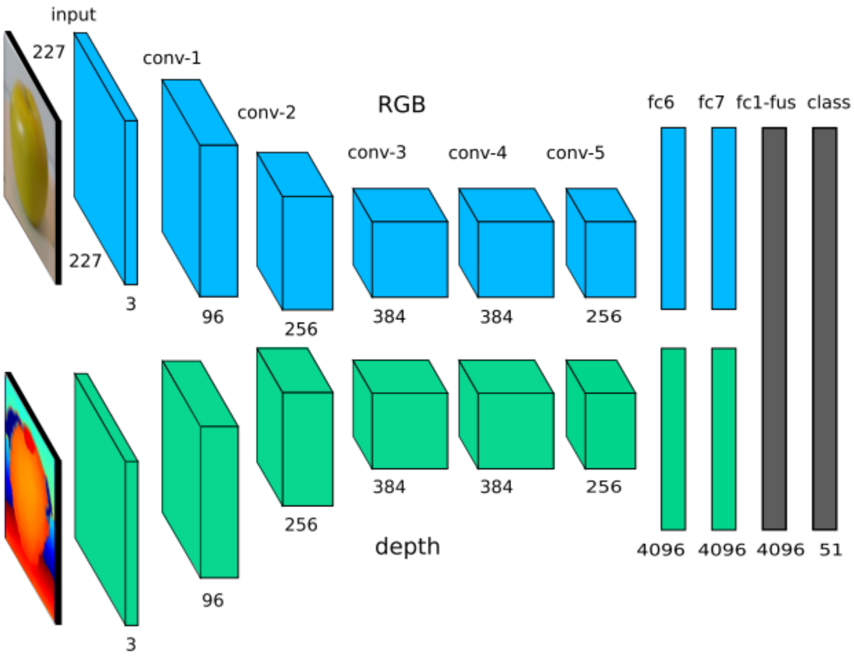


Figure 4: Model architecture proposed by Eitel et al. [1].

Figure 5 shows our model, constructed by following the description in Figure 4. The model is plotted using visualizer package of Keras. Using Keras, each stream is constructed using `Sequential()` model and the fusion layer is created by `Merge()` layer. We propose to augment each convolutional layer (`Convolution2D()`) with a maxpooling (`MaxPolling2D()`) and a dense sampling layer (`Dense()`) to reduce the risk of over-fitting and decrease memory usage.

In fact, there are two different training process: one to find the weights of stream model and one for the fusion model. To train a stream model, we create an architecture similar to a branch in Figure 5 and add one classifier at the end (Figure 6). The weights of trained streams are reused to initialize the two branches of the whole network.

5.3 Other accomplishments

Beside data preprocessing and model construction, we have also successfully configured a GPU environment for Keras/Theano to speed up training process. In addition, the paper mentions about using CaffeNet to initialize the network. However, this pretrained network has to be converted to Keras format before loading. We utilize a modified version of Keras that includes conversion package, provided by Solà [4].

6 Current reservations and questions

Model implementation is taking longer than expected with respect to getting our code to run on the GPU, but otherwise, we are on track.

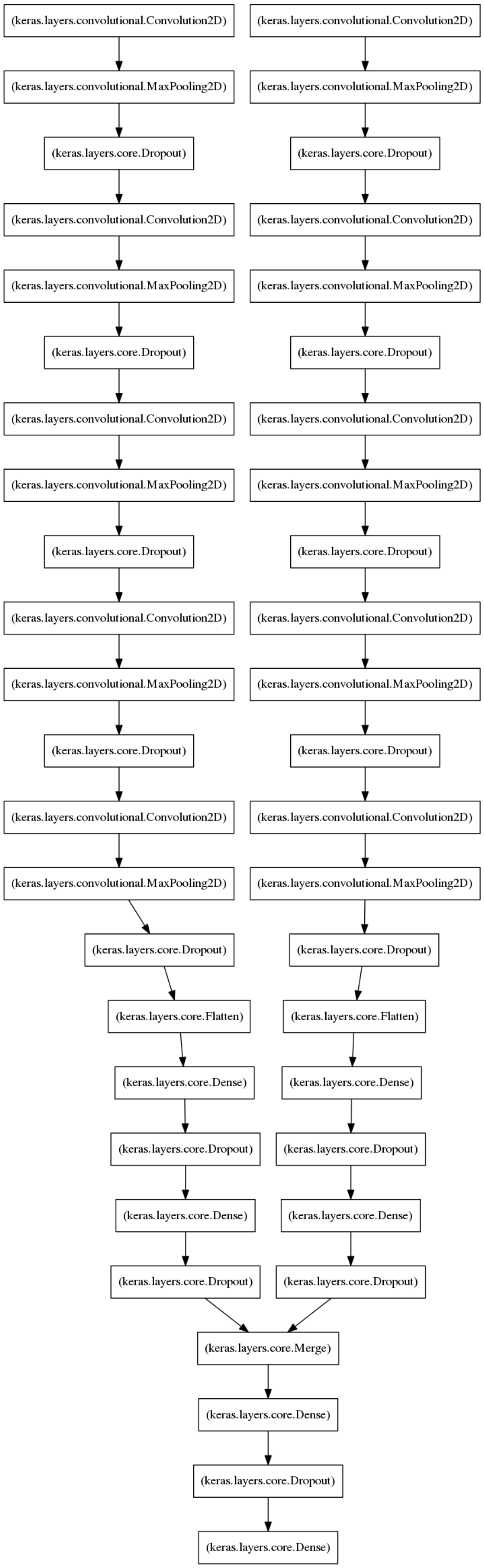

(keras.layers.convolutional.Convolution2D)
(keras.layers.convolutional.Convolution2D)
(keras.layers.convolutional.MaxPooling2D)
(keras.layers.convolutional.MaxPooling2D)
(keras.layers.core.Dropout)
(keras.layers.core.Dropout)
(keras.layers.convolutional.Convolution2D)
(keras.layers.convolutional.Convolution2D)
(keras.layers.convolutional.MaxPooling2D)
(keras.layers.convolutional.MaxPooling2D)
(keras.layers.core.Dropout)
(keras.layers.core.Dropout)
(keras.layers.convolutional.Convolution2D)
(keras.layers.convolutional.Convolution2D)
(keras.layers.convolutional.MaxPooling2D)
(keras.layers.convolutional.MaxPooling2D)
(keras.layers.core.Dropout)
(keras.layers.core.Dropout)
(keras.layers.convolutional.Convolution2D)
(keras.layers.convolutional.Convolution2D)
(keras.layers.convolutional.MaxPooling2D)
(keras.layers.convolutional.MaxPooling2D)
(keras.layers.core.Dropout)
(keras.layers.core.Dropout)
(keras.layers.convolutional.Convolution2D)
(keras.layers.convolutional.Convolution2D)
(keras.layers.convolutional.MaxPooling2D)
(keras.layers.convolutional.MaxPooling2D)
(keras.layers.core.Dropout)
(keras.layers.core.Dropout)
(keras.layers.core.Flatten)
(keras.layers.core.Flatten)
(keras.layers.core.Dense)
(keras.layers.core.Dense)
(keras.layers.core.Dropout)
(keras.layers.core.Dropout)
(keras.layers.core.Dense)
(keras.layers.core.Dense)
(keras.layers.core.Dropout)
(keras.layers.core.Dropout)
(keras.layers.core.Merge)
(keras.layers.core.Dense)
(keras.layers.core.Dropout)
(keras.layers.core.Dense)

Figure 5: Fusion model with RGB and depth streams.

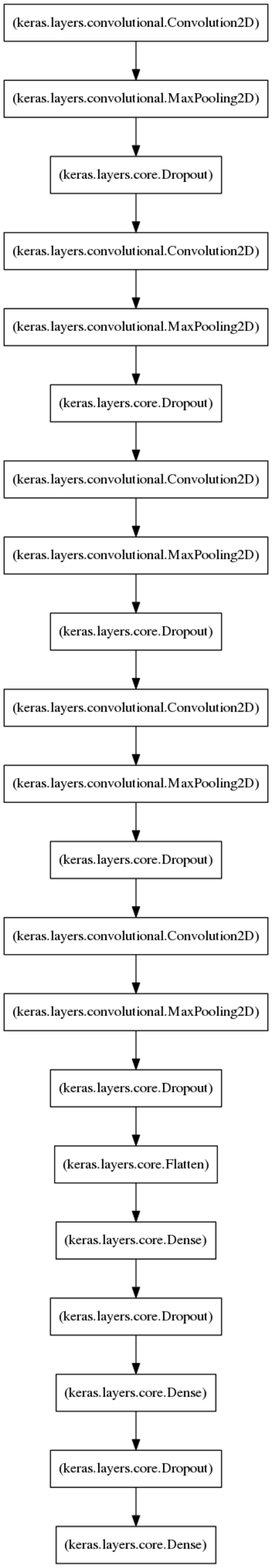

(keras.layers.convolutional.Convolution2D)
(keras.layers.convolutional.MaxPooling2D)
(keras.layers.core.Dropout)
(keras.layers.convolutional.Convolution2D)
(keras.layers.convolutional.MaxPooling2D)
(keras.layers.core.Dropout)
(keras.layers.convolutional.Convolution2D)
(keras.layers.convolutional.MaxPooling2D)
(keras.layers.core.Dropout)
(keras.layers.convolutional.Convolution2D)
(keras.layers.convolutional.MaxPooling2D)
(keras.layers.core.Dropout)
(keras.layers.convolutional.Convolution2D)
(keras.layers.convolutional.MaxPooling2D)
(keras.layers.core.Dropout)
(keras.layers.core.Flatten)
(keras.layers.core.Dense)
(keras.layers.core.Dropout)
(keras.layers.core.Dense)
(keras.layers.core.Dropout)
(keras.layers.core.Dense)

Figure 6: Single stream model.

References

[1] A. Eitel, J. T. Springenberg, L. Spinello, M. A. Riedmiller, and W. Burgard, "Multimodal deep learning for robust RGB-D object recognition," *CoRR*, vol. abs/1507.06821, 2015.

[2] Keras, "Examples - keras documentation." `http://keras.io/examples/`. (visited on 2016-April-10).

[3] K. Lai, L. Bo, X. Ren, and D. Fox, "A large-scale hierarchical multi-view RGB-D object dataset," in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pp. 1817–1824, May 2011.

[4] M. B. nos Solà, "Deep learning library for Python. Convnets, recurrent neural networks, and more. Runs on Theano and TensorFlow." `https://github.com/MarcBS/keras`. (visited on 2016-April-09).