RGB-D Object Recognition using Deep Learning

Dario Aranguiz
University of Illinois at Urbana-Champaign
arangui2@illinois.edu

Cu-Khoi-Nguyen Mac
University of Illinois at Urbana-Champaign
knmac@illinois.edu

Abstract

TODO

1 Introduction

Define and motivate the problem, discuss background material or related work, and briefly summarize your approach.

Photography has remained generally unchanged since the first photograph was taken in 1816 [1]. The proceeding years since have seen a plethora of technical advances bettering image quality and capture rate, but imaging has remained a two-dimensional art. Developments in recent years, however, have begun to fundamentally alter how we consider imaging, with the release of the first Xbox Kinect in November 2010 [2]. With this launch, consumers and researchers alike were able to view the world in three dimensions, opening up a second modality to image processing. Our particular interest in this area is to explore how object recognition pipelines can be changed to accommodate this new depth information.

Before the advent of deep learning, object classification was done using a process called *bag of features*. At a high level, features are extracted from an image and clustered in some manner such that every feature in a particular image belongs to a cluster. This is called a "visual vocabulary". Finally, objects are quantized by forming a histogram of visual codeword frequency; as an example, a bicycle may have a high frequency of codewords corresponding to spokes on the wheels, but it may have a low frequency of codewords corresponding to the hubcaps of a car. New objects can then be classified using a nearest-neighbor approach, where neighbors are high- dimensional cluster centroids of known objects, or using other standard classification techniques. This was done most notably by Lazebnik et al in 2006, achieving a maximum classification rate of 64.6% on the Caltech101 image dataset [todo ref].

Deep learning has since supplanted bag of features as the state-of-the-art in image classification, with the largest paradigm shift being seen in 2012 with the publication of AlexNet, a network trained for the ImageNet classification challenge [todo ref]. There have been numerous network architectures since AlexNet, such as VGG Net [todo ref], GoogLeNet [todo ref], and ResNet, a deep residual network with 152 layers and only 3.57% error on the ImageNet test set [todo ref].

Our approach combines the contributions of AlexNet with new depth sensing by converting depth information into an RGB image and separately classifying objects with both their original RGB information as well as their depth information. This classification is then fused and recomputed to produce a combined classification incorporating both depth and color.

2 Details of approach

Include any formulas, pseudocode, diagrams – anything that is necessary to clearly explain your system and what you have done. If possible, illustrate the intermediate stages of your approach with result images.

For our approach, we implemented a variation on the CVPR 2015 paper from Eitel et al titled *Multimodal Deep Learning for Robust RGB-D Object Recognition* [3]. The main contribution of this paper was to show how pretrained networks such as AlexNet or VGGNet can be used to process single-channel depth information, enabling networks to run on depth information despite a lack of large-scale depth datasets such as ImageNet for color images [4]. In this section, we will discuss our network architecture, our preprocessing phase, and our training methodology.

2.1 Dataset

In this project, we use Washington RGB-D object dataset [5] . This dataset includes 300 common household object, divided into 51 categories (Figure 1). Using a Kinect style 3D camera, data are recorded, synchronized, and aligned at the framerate of 30Hz and resolution of 640 x 480. Each object is recorded while being placed on a turntable under three different camera viewpoints: low, middle, and high.

Figure 1: Some samples from RGB-D Object Dataset.

The dataset is structured as follow:

```
<category>/<object_id>/<sample_id>
```

where

```
<object_id> = <category>_<number>
```

and

```
<sample_id> = <category>_<number>_<video>_<frame>
```

For example

```
banana/banana_1/banana_1_2_5
```

means that the sample is the fifth frame of the second video sequence (out of three different camera viewpoints). This sample records object `banana_1`, which is the first object in `banana` category. Each video sequence contains around 150 samples.

For each sample, there are four different kinds of data: color image (Figure 2a), depth image (Figure 2b), segmentation mask (Figure 3), and a text file indicating the top-left corner of the object of interest.

(a) Color image (b) Depth image

Figure 2: Original color and depth photos of object captured on a turntable.

Figure 3: Ground-truth segmentation mask for banana object.

2.2 Network Architecture

The network structure proposed by Eitel et al [3] is pictured in Figure 4. Its structure can be broken into three parts: a network for color-based classification, a network for depth-based classification, and a final fusion network combining the two single-modality streams into a final classification result. Fusing two single-stream networks is beneficial for two main reasons. Firstly, unlike large companies like Microsoft and Google who have recently become involved in deep learning competitions, we do not have a GPU farm at our disposal to train networks ad-infinitum. Additionally, the first consumer depth camera was only released within the last 5 years [2]. Unlike ImageNet, we do not have millions of labeled depth captures to use for network training. A smaller dataset could be used to train a depth network from scratch, but using such a comparatively small dataset would likely result in network overfitting and poor results on test data.

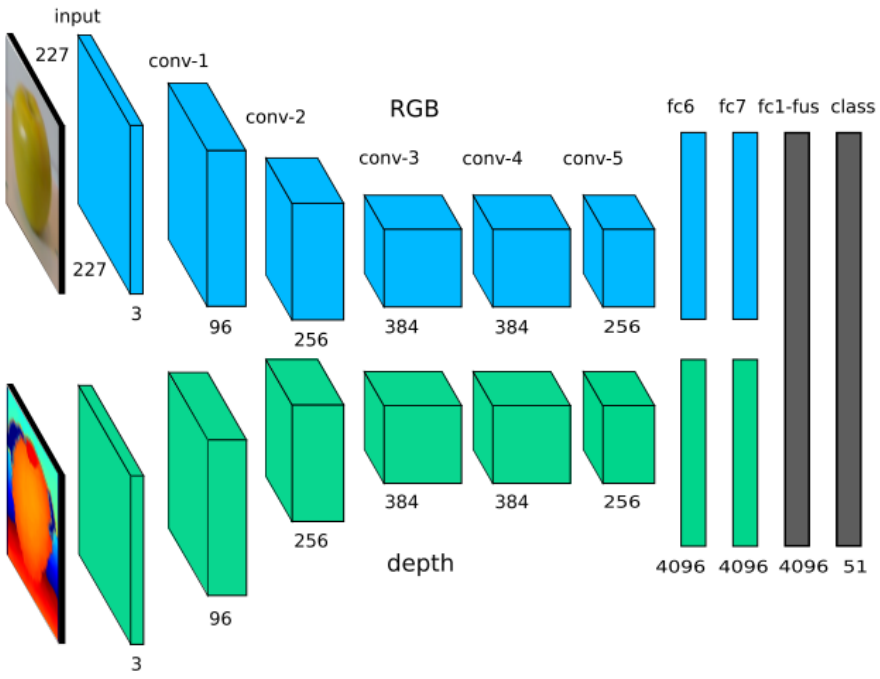


Figure 4: Network architecture using separate color and depth inputs. Inputs for color and depth are $227 \times 227 \times 3$, assuming depth colorization during preprocessing stage. Each modality has its own individual network, with the two resultant fusion layers providing final classification.

To this end, we use the pretrained network AlexNet for our single stream network. AlexNet is a top-tier network without as many layers as a deep residual learning network, making it a better choice for our limited compute capacity and timeframe. Our implementation of AlexNet is shown in further detail in Figure 5.

Finally, we strip the `fc8` classification layers from the our AlexNet streams and concatenate the two models, feeding the now 8092-node layer into `fc1-fus`, a 4096-node fully-connected layer. This then goes into a final classification layer with softmax activation. The individual elements of the network are described below.

2.2.1 Convolutional Layer

TODO, what is a convolutional layer

2.2.2 ReLU

TODO, what is ReLU

- `conv-1`:
  - $96 \times 11 \times 11$ Convolutional Filter
  - ReLU Activation
  - Batch Normalization
  - 2×2 Max Pooling
- `conv-2`:
  - 2-width Zero Padding
  - $256 \times 5 \times 5$ Convolutional Filter
  - ReLU Activation
  - Batch Normalization
  - 2×2 Max Pooling
- `conv-3`:
  - 1-width Zero Padding
  - $384 \times 3 \times 3$ Convolutional Filter
  - ReLU Activation
- `conv-4`:
  - 1-width Zero Padding
  - $384 \times 3 \times 3$ Convolutional Filter
  - ReLU Activation
- `conv-5`:
  - 1-width Zero Padding
  - $256 \times 3 \times 3$ Convolutional Filter
  - ReLU Activation
  - 2×2 Max Pooling
- `fc6`:
  - 4096-node Fully Connected Layer
  - ReLU Activation
  - 50% Dropout
- `fc7`:
  - 4096-node Fully Connected Layer
  - ReLU Activation
  - 50% Dropout
- `fc8`:
  - 51-node Fully Connected Layer
  - Softmax Activation

Figure 5: Single-stream network layout for depth and color channels respectively. Dropout layers are removed during test time.

2.2.3 Batch Normalization

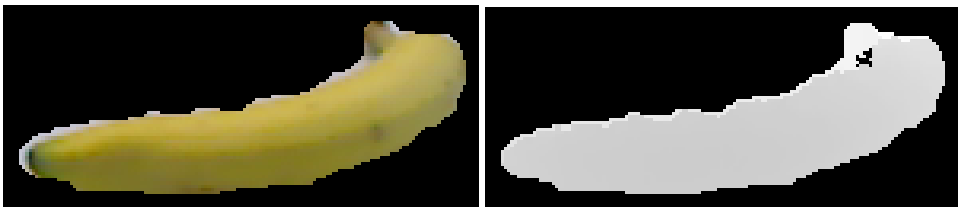
TODO, what is Batch Normalization

2.2.4 Max Pooling

TODO, what is max pooling

2.3 Preprocessing

Preprocessing for this network has three primary stages: segmentation, rescaling, and depth colorization. First, we segment the color and depth images using a mask provided with the dataset, setting pixel values to black in the color image and zero in the depth image (Figure 6). Our dataset provides segmentation masks for every object capture in the dataset, although Eitel et al [3] showed in their paper that doing classification without the segmentation mask was also possible.



(a) Color image (b) Depth image

Figure 6: Cropped banana photo using given segmentation mask.

The image pairs are then cropped and rescaled to 227×227, the size of the input for the single-stream networks. We do this by scaling the image by a factor:

$$\kappa = \frac{\max(W, H)}{227}. \tag{1}$$

where W and H are object's width and height, respectively. The smaller dimension is then zero-padded to fit the square 227×227. By this way the object is kept as the image's center, making it efficient in case the boundary is cropped during training (Figure 7a). More interesting, however, is the process of depth colorization ((Figure 7b)). Depth sensors like the Xbox Kinect only give a single-channel intensity image proportional to the distance from the sensor. By converting depth images into three-channel data, we can treat them as images and feed into AlexNet (or other pre-trained image recognition networks). Although there are different ways to colorize depth maps, jet colorization is proven superior in term of boosting system's performance [3].

2.4 Network Training

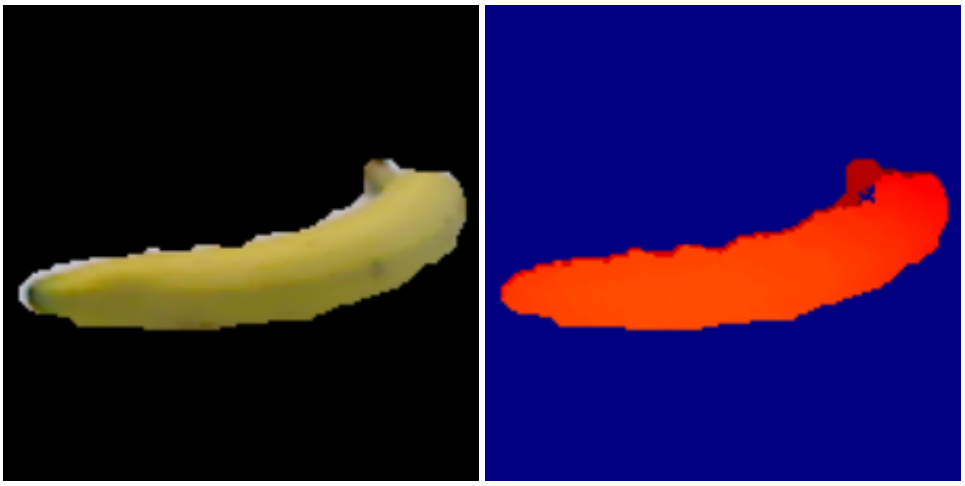
The training process in divided into two different phases: (1) training the stream networks and (2) training the fusion network. Suppose that we have the dataset

$$\mathcal{D} = \left\{ \left(\mathbf{x}^1, \mathbf{d}^1, \mathbf{y}^1 \right), ..., \left(\mathbf{x}^N, \mathbf{d}^N, \mathbf{y}^N \right) \right\} \tag{2}$$

where $\mathbf{x}^i$ is a RGB image, $\mathbf{d}^i$ is a depth image, and $\mathbf{y}^i$ is a label in the form of

$$\mathbf{y}^i = (y_1^i, y_2^i, ..., y_k^i, ..., y_M^i)^\top \qquad \text{s.t.} \qquad y_k^i = \begin{cases} 1, & \mathbf{x}^i \text{ and } \mathbf{d}^i \text{ belong to class } k \\ 0, & \text{otherwise} \end{cases} \tag{3}$$

where M is the number of classes. The data are fed into the first training phase and then the second one.



(a) Color image (b) Depth image

Figure 7: Rescaled cropped color photo and rescaled cropped depth photo colorized using *Jet* heatmap.

2.4.1 Training the stream networks

Let $g^I(\mathbf{x}^i;\theta^I)$ be the representation for the color image $\mathbf{x}^i$ of the last fully connected layer from stream networks (*fc7*) of AlexNet, where θ^I is the parameter. Similarly, we have $g^D(\mathbf{d}^i;\theta^D)$ for the depth image $\mathbf{d}^i$. Since we initialize the stream networks with pretrained weights, θ^I and θ^D are known. The weights $\mathbf{W}^I$ and $\mathbf{W}^D$ for RGB and depth streams are trained by solving

$$\min_{\mathbf{W}^I,\theta^I} = \sum_{i=1}^{N} \mathcal{L}\left(\text{softmax}\left(\mathbf{W}^I g^I\left(\mathbf{x}^i;\theta^I\right),\mathbf{y}^i\right)\right), \tag{4}$$

$$\min_{\mathbf{W}^D,\theta^D} = \sum_{i=1}^{N} \mathcal{L}\left(\text{softmax}\left(\mathbf{W}^D g^D\left(\mathbf{d}^i;\theta^D\right),\mathbf{y}^i\right)\right), \tag{5}$$

where softmax function is

$$\text{softmax}(z) = \frac{e^z}{\|z\|_1} \tag{6}$$

and the loss is

$$\mathcal{L}(x,y) = -\sum_k y_k \log s_k. \tag{7}$$

This loss function is actually the "categorical crossentropy" function, which is commonly used in neural networks.

2.4.2 Training the fusion network

After acquiring $\mathbf{W}^I$ and $\mathbf{W}^D$, we discard the softmax layers (*fc8*) and concatenate the last responses of the two streams: $g^I(\mathbf{x}^i;\theta^I)$ and $g^D(\mathbf{d}^i;\theta^D)$ and feed them through the additional fusion layer (*fc1_fus*)

$$\mathcal{F} = f\left(\left[g^I(\mathbf{x}^i;\theta^I); g^D(\mathbf{d}^i;\theta^D)\right];\theta^F\right) \tag{8}$$

where θ^F is the parameters of this layer. We can train this layer with a similar manner as in the previous training phase:

$$\min_{\mathbf{W}^F,\theta^I,\theta^D,\theta^F} = \sum_{i=1}^{N} \mathcal{L}\left(\text{softmax}\left(\mathbf{W}^F\mathcal{F},\mathbf{y}^i\right)\right) \tag{9}$$

Note that in this phase, the weights trained from the previous one are kept unchanged. Only the weights of the fusion network are optimized.

3 Results

Clearly describe your experimental protocols. If you are using training and test data, report the numbers of training and test images. Be sure to include example output figures. Quantitative evaluation is always a big plus (if applicable). If you are working with videos, put example output on YouTube or some other external repository and include links in your report.

3.1 Framework

This project is implemented based on Keras framework [6], which is a modular neural network library. The library is written in Python and can run on top of Theano or TensorFlow. In this project, we propose to use Theano backend [7] for implementation.

Since we need to use pretrained weights from AlexNet, our system has to be able to load this network. However, AlexNet is provided under caffemodel format, which is used in CaffeNet-based deep learning networks [8]. The problem is original Keras framework does not allow to load caffemodel directly. Therefore, we use a modified version of Keras, provided by Solà [9]. This version includes a fully functional Keras framework with an additional package for converting from caffemodel to keras format.

3.2 Implementation

The project contains 4 main modules: data processor, model architecture, model training, and model testing.

3.2.1 Data Processor

The module takes care of loading and preprocessing input data. The input data are cropped with respect to the ground truth segmentation mask beforehand, as showed in Figure 6. For preprocessing task, inputs undergo the procedure introduced in Section 2.3: images are resized efficiently to the size of 227x227 (based on Equation 1) and depth maps have an additional step of colorizing.

For data loading, the module receive a batch of image, a dictionary list (containing all possible categories), and location of data. Since the amount of data is significant, we cannot load everything at the same time, therefore batch processing is necessary. Based one the naming format of each sample in the batch (Section 2.1), we can easily figure out which category that sample belong to and construct the label vector according to the given dictionary as seen in Equation 3. Since we are using pretrained weights from AlexNet to train the network, it is essential to remove the mean image [1] of ImageNet from every input after loading.

3.2.2 Model Architecture

This module creates the architecture for the stream and fusion networks. Common Keras-based systems are implemented based on Sequential model, which allows adding layers in a sequential manner. Such scheme of architecture is actually found in popular network, such as AlexNet [10], VGG [11]. However, such architecture faces the problem of constructing fusion layer. Although Keras provides layer called Merge layer to concatenate different sources of input, it does not work well with Sequential model, making it impossible to load networks after training. To overcome this problem, we use Graph models instead, which offers more flexibility but still retains core functionalities as in Sequential models.

The module has two main parts: the first one replicates the network architecture of AlexNet and the second one constructs the fusion network from stream models. The detail of stream model is illustrated in Figure 5. Note that we use the same network architecture for both color and depth inputs. Since Keras' Graph models are heavily based on naming convention (each layer is represented as a node and the relationship between different nodes is defined using their names), we include a tag name, being `_rgb` or `_dep`. Using those tags, we can distinguish the color from depth streams while

[1] A mean image for AlexNet under NumPy format is available at: `https://github.com/BVLC/caffe/blob/master/python/caffe/imagenet/ilsvrc_2012_mean.npy`.

fusing. For the second part, the module receives color and depth stream (already trained) as inputs and duplicates from layer *fc1* to *fc7* and add parameter `trainable = False` for convolutional and normalization layers. This helps freezing the trained weights of those layers, making the fusion network focus on training fusion layer only.

After merging outputs from stream models and dense sampling with 4096 nodes, we add another Dense sampling layer that produces 51 nodes, corresponding to 51 categories (as seen in Figure 4), for classification. This classification layer requires `softmax` function as activation. In Keras, there are two ways to declare activation for a layer. The first way is to define activation while adding the node

```
model.add_node(Dense(nb_classes, activation='softmax'), \
               name='softmax_fus', input='fc1_fus')
model.add_output(name='output', input='softmax_fus')
```

The second way is to refactor the activation function as a separate node

```
model.add_node(Dense(nb_classes), name='fc2_fus', input='fc1_fus')
model.add_node(Activation('softmax'), name='softmax_fus', \
               input='fc2_fus')
model.add_output(name='output', input='softmax_fus')
```

Although they look similar, the difference actually affect the results significantly. A more detailed explanation on its effect is presented in Section 3.3.

3.2.3 Model Training

This module trains all models, including color stream, depth stream, and fusion network. Stream models are trained first then the fusion one. After finishing training, a model is saved for future use as two different files: a `.json` file to store a json query indicating model's structure and a `.h5` to store model's weights under HDF5 format. This process is done easily by Keras' provided functions: `to_json()` and `save_weights()`.

Training process is conducted as several epochs. For each epoch, the training samples are shuffled once to guarantee network's generosity. After training, the model is evaluated on evaluation samples to compute its loss and prediction accuracy. The system is equipped with an early stopping mechanism: if there is no improvement in accuracy (or if the improvement is too insignificant) for K continuous epochs, the training process is halted. The value K is called training's `patience` and is set as 10.

3.2.4 Model Testing

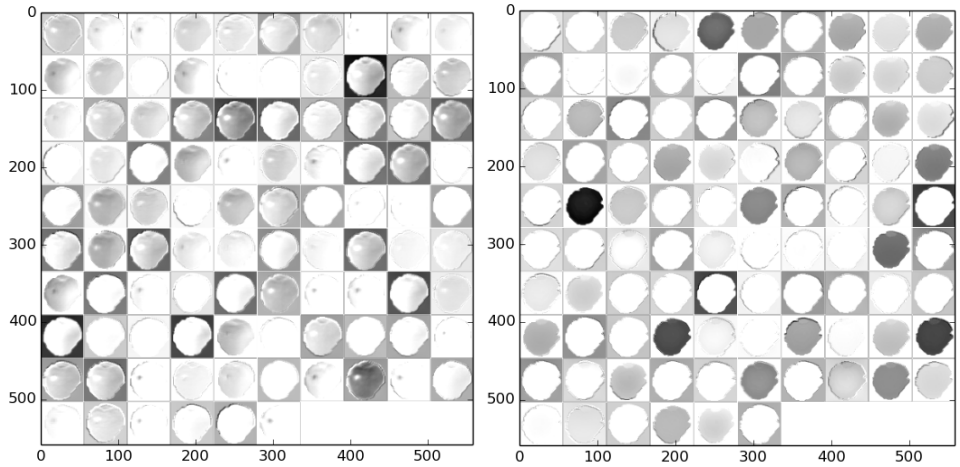
The testing process is similar to evaluation. In this module, the trained model is tested on testing samples, which are separated from training and evaluation set. Models are loaded using Keras' functions `model_from_json()` and `load_weights()`. To fully analyze system's performance, the module tests prediction accuracy on all three models: color, depth, and fusion. Detailed results are provided in Section 3.3.

3.3 Experimental Results

To conduct the experiment, we divide RGB-D object data set into three different parts, corresponding to training, evaluation, and testing. As showed in Section 2.1, each category contains different objects with some specific ID. For every category, we randomly choose an object for evaluation and another one for testing, while the rest is used for training. Since the number of objects varies for every category, the amount of training object also differs. The generated samples for training, evaluation, and testing are saved as lists for later reuse.

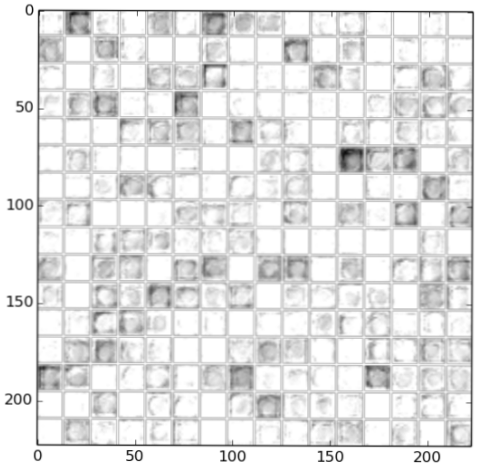
Figure 8 displays intermediate outputs at the first and fifth activation layers (using ReLu function), given an apple as input. Figure 8a and 8c are the visualizations of color image while Figure 8b and 8c are those of depth. It is clear that the object's shape is retained on both channels and the features get

more complicated at deeper layers. Despite adding artificial information (depth colorizing), things we would expect to see like edge and magnitude detection are still present in the activations.

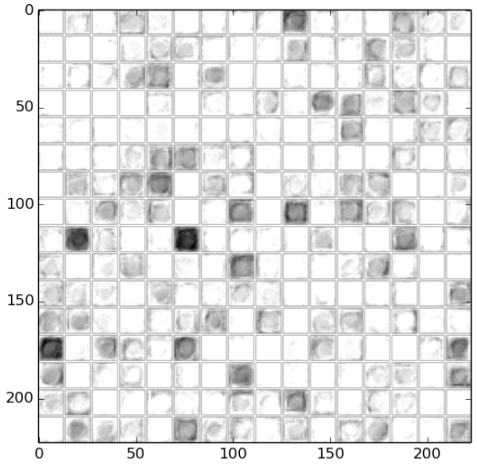


(a) Activation of ReLU1 layer for RGB image

(b) Activation of ReLU1 layer for Depth image



(c) Activation of ReLU5 layer for RGB image



(d) Activation of ReLU5 layer for Depth image

Figure 8: Visualization of intermediate results on the first and fifth ReLU layers on both RGB and depth images of an apple.

Prediction accuracy of the system is presented in Table 1. We conduct experiences in three different scenarios: reduced data (with 5 classes), full dataset (with 51 classes), and full dataset with refactored fusion layers. Accuracy on stream models are satisfactory and stable, around 68% for RGB stream and 79% for depth stream. Fusion network, however, has poorer accuracy. Although it is 56.52% on reduced dataset, the accuracy drops to 8.07% on the full one.

Table 1: Recognition accuracy on reduced (5 categories) and full dataset (51 categories) for RGB stream, depth stream, and fusion network

	RGB stream	Depth stream	RGB-D fusion network
Reduced dataset	68.26%	74.43%	56.52%
Full dataset	68.93%	79.19%	8.07%
Full dataset with refactored fusion layers	68.93%	79.19%	17.34%

The effect is illustrated with better insight in Figure 9, where prediction is computed on 200 samples randomly selected from testing set. As the figure shows, most of the samples are classified into two classes although the groundtruth labels are well distributed among different categories. However, if

we use refactored script for fusion layers, the prediction is better distributed, as showed in Figure 10. Therefore, we suspect that the problem is caused by some API bug in Keras framework.

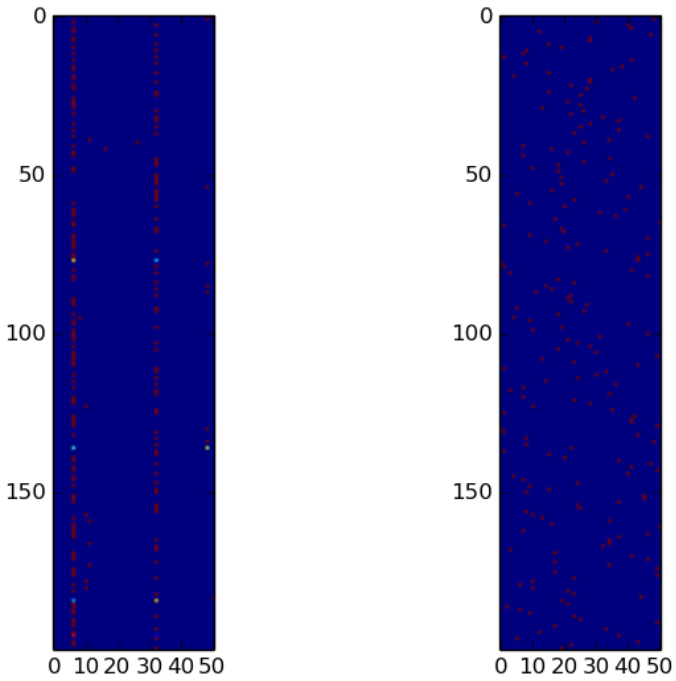


Figure 9: Prediction results (left) and ground truth labels (right) on 200 random samples with unfactored fusion layer.

4 Discussion and conclusions

Summarize the main insights drawn from your analysis and experiments. You can get a good project grade with mostly negative results, as long as you show evidence of extensive exploration, thoughtfully analyze the causes of your negative results, and discuss potential solutions.

5 Individual contributions

Required if there is more than one group member.

References

[1] B. Newhall, *The History of Photography*. New York: The Museum of Modern Art, 1982.

[2] A. Pham, "E3: Microsoft shows off gesture control technology for xbox 360." `http://latimesblogs.latimes.com/technology/2009/06/microsofte3.html`, June 2009. (visited on 2016-May-10).

[3] 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.

[4] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge," *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015.

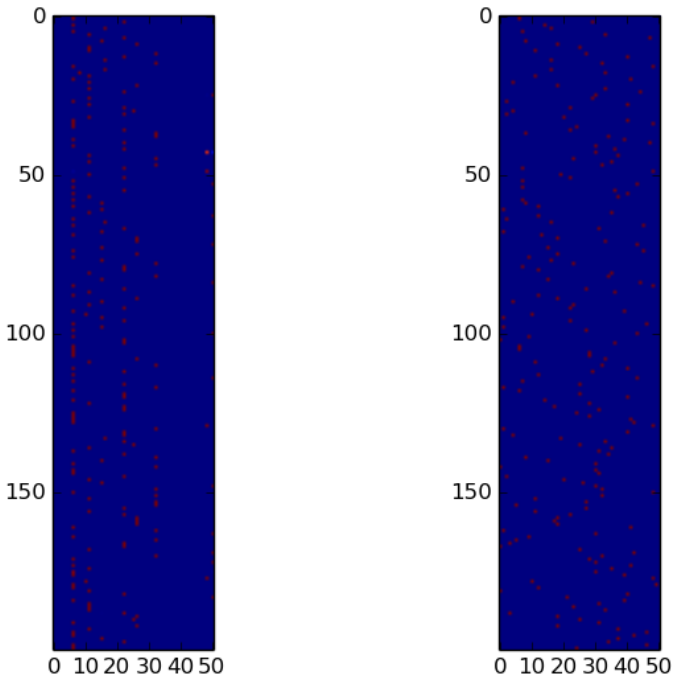


Figure 10: Prediction results (left) and ground truth labels (right) on 200 random samples with refactored fusion layer.

[5] 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.

[6] F. Chollet, "keras." `https://github.com/fchollet/keras`, 2015.

[7] Theano Development Team, "Theano: A Python framework for fast computation of mathematical expressions," *arXiv e-prints*, vol. abs/1605.02688, May 2016.

[8] 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.

[9] 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).

[10] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems 25* (F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, eds.), pp. 1097–1105, Curran Associates, Inc., 2012.

[11] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *CoRR*, vol. abs/1409.1556, 2014.