Object state recognition for learning manipulation tasks in robotics

Sameera Horawalavithana

Abstract— **Object state recognition is an open problem which is a special version of object detection problem. In the field of computer vision, this problem has strong roots in object-object interaction affordance learning of robots [6]. We model this problem as a multi-class classification problem given a manually annotated dataset representing the states of 24 cooking objects. We present a modified *ConvNet* architecture with the bottom layers trained using ImageNet examples. Our dataset is used to train top layers in the *ConvNet* on classifying the correct motion state of cooking objects. We achieve 74% accuracy of predictions for unseen examples, where it generalizes well on a relatively small training dataset of 5000 examples.**

I. INTRODUCTION

Object detection and recognition techniques are widely popular in multiple application domains. As an example, pedestrian detection techniques are used in many surveillance systems. Together with face recognition techniques, such applications can be further improved for object verification tasks. Hence, the recognition of objects plays an important role in many computer vision problems.

In our work, we focus on a special version of object recognition problem called **object state recognition**. Here, we assume objects are functionally oriented, such that we can observe different states of the same object in multiple time intervals [3]. Importantly, we consider objects are functionally related in a single or shared domains [5]. The states are drawn from a finite space which is local to the domain. Such that, different objects can belong into the same state.

In a motivational example, we present a series of object manipulation tasks performed by a *chef robot*. Given an initial state of an object, *chef robot*'s task is to reach the given end state, performing a sequential steps of operations. Such steps are granular and usually conditioned on the temporal state of the objects. As an example, given a whole piece of carrot, the first step would be to peel. Then the peeled carrot would be grated using a downward motion. Finally, you might need to discard the smallest piece that you can not shred longer. This operation includes carrot as an object which needs to be detected first. Existing object detection techniques or manual entry of such knowledge to the robots can be considered. The steps are conditioned by the states of whole, peeled, grated and other. *chef robot* requires to identify the object in state to perform correct steps (e.g., grate the peeled carrot). Any mis-identification leads to incorrect order of steps, that may end with a wrong final state. Sometimes, such mistakes can not be recovered (e.g. if whole carrot is grated before peeled.)

In general, cooking objects are functionally related such that you apply a same set of functions to reach a shared set of states (e.g., whole, diced, grated etc.). Such functions could be in different order based on the associated cooking objects (e.g., peel both carrot and potato, and slice potato before grating carrot). As *chef robot* learn functions based on instructional videos, it's important to provide the states of the objects as the knowledge base. We use *Convolutional Neural Networks (CNN)* architecture for such learning.

The remainder of this paper is organized as follows. In Section II, we introduce our dataset, while we present CNN architecture in Section III. We present our results in Section IV and discuss our contributions in Section V.

II. DATA AND PREPROCESSING

The dataset include 5100 image examples which has 24 cooking objects in 7 states (i.e., whole, diced, sliced, grated, julienne, juiced, creamy). We manually annotate objects in a participation of 20 human experts. Each object is assigned only one state. We select the most discriminative state for images that have multiple states, and categorize images that have multiple objects in a separate class of interest. Table I presents the distribution of the dataset over all possible states that we consider, while a sample image of a cooking object is presented in Figure 1.

state	number of examples
diced	605
jullienne	408
sliced	1103
grated	700
whole	1117
juiced	546
creamy_paste	638

TABLE I: Basic statistic about the dataset

We randomly select 20% of examples for validation data similar to the distribution of training data, and have a separate set for testing. Both training and validation data are shuffled randomly. We apply several image pre-processing techniques to get a set of augmented images for training, validation and testing. Such pre-processing steps include:

- image rescale: transform pixel values to a probability in RGB scale.
- feature centralization: normalize features into a Gaussian distribution with mean 0.
- feature normalization: divide pixel densities by the standard deviation of feature values.
- feature shift: horizontal and vertical shift of feature values randomly

Fig. 1: Sample object: **Onion** with the state *whole*

- histogram equalization: adjust contrast of the images separately for RGB color values[1].
- zoom and shear images
- random rotation of images

All images are resized to 512×512 by keeping the original aspect ratio.

III. Methodology

Our method follows *convolutional networks (ConvNet)*: deep learning models that achieved a great success in object recognition tasks. Recall that our objective is to identify correct states in cooking objects. We experiment with VGG model [4] which introduced 16-19 depth of weight layers in ConvNet (Section III-A). We improve VGG model by adding an extra convolutional layer followed by three fully connected layers with different number of channels (Section III-B). Further, we describe the trial-and-error process on improving prediction results (Section III-C)

A. VGG

We use the 16-layer VGG ConvNet model which is used to classify images in recent ImageNet competition. VGG includes a stack of convolutional layers followed by three fully connected layers at the end. We only use bottom layers of VGG, excluding fully connected layers such that we can feed arbitrary sizes of images. VGG uses 3×3 filters as receptive fields, and the number of filters vary over 64 to 512. Maxpooling has been used as a method to sub-sample in 5 convolutional blocks.

B. Our configuration

Figure 2 presents a summarized version of the complete ConvNet model that we used. For training, the input to our ConvNet is a set of 512×512 RGB images. Images are pre-processed prior to training as presented in Section II. The initial model presents the set of frozen VGG layers with the weights trained from ImageNet.

[1]https://en.wikipedia.org/wiki/Histogram_equalization

1) Convolutional Layers: The task of convolutional layer is to extract features for learning hidden patterns of images. Since the original images are passed to VGG model, its' convolutional layers learn most direct features from the images. Figures 3a to 3h visualize the VGG's first block of convolutional layer with the filtered images of two convolutional layers that have 64 filters each. Note that we only present the visual space of first four filters. They mostly learn the direct features of direction and color of the images, as the original shape preserved in the filtered images.

More convolutional steps will only break down into more complicated features using the knowledge from previous layers. As an example, Figure 4 visualizes the filtered images at the top convolutional layers close to the output layer. More complicated textures are presented by Figures 4i to 4p which is the extra convolutional block after VGG layers. It's clear that these textures are more complicated than the filtered images output in the convolutional layers in VGG block 05 (Figures 4a to 4h). These complicated feature maps represent grid and spot textures which capture more hidden information of a larger spatial context.

After few steps of experiments, we limit to the addition of single convolutional layer after initial VGG model. We experimented with two to three convolutional layers with a decreasing magnitude of filters, but the significance of accuracy remains in a smaller range.

Similar to VGG, we also use 3×3 filters as receptive fields with the padding outlines the output in same length compared with the input. We set the number of filters to 512 in our convolutional layer, which proves to be the optimal number of filters. This decision is also motivated to have a number of filters similar to the final convolutional layer in VGG. All activation layers immediately followed by convolutional layers are equipped with ReLU units.

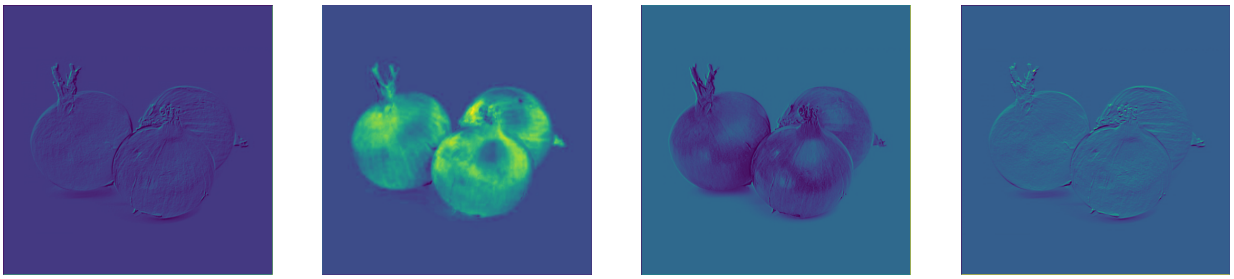
2) Pooling Layers: Pooling layers are used to subsample feature maps to have better spatial invariance. We apply max pool operation with a 2×2 pool size, and a similar size of strides. Such that, image would be reduced half by size in each dimensions by taking the maximum value of each pool.

3) Normalization Layers: We only apply batch normalization after each convolutional and fully connected layers. The goal is to have a normal distribution over the activation results of previous layers, since layers are susceptible to the changes of input of previous layers. Batch normalization avoid this problem of internal covariate shift [2].

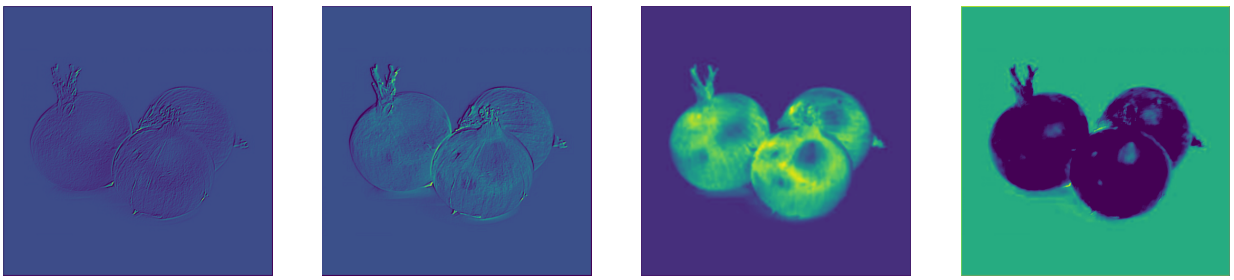
4) Dropout Layers: We note the invariance of relatively small amount of training data, and the large number of parameters to train. The probability of dropout individual neurons is set to 0.5. We reduce the number of activation by half with this way of regularization. A dropout has been applied in the convolutional block. The main objective is to reduce the number of parameters and avoid over-fitting.

We try with the addition of dropout units after fully connected layers with relatively larger in size. Later, we avoid dropout and reduce the size of neurons in the fully connected layers. Our concern was the random drop of hidden units without an optimal way of selecting which to

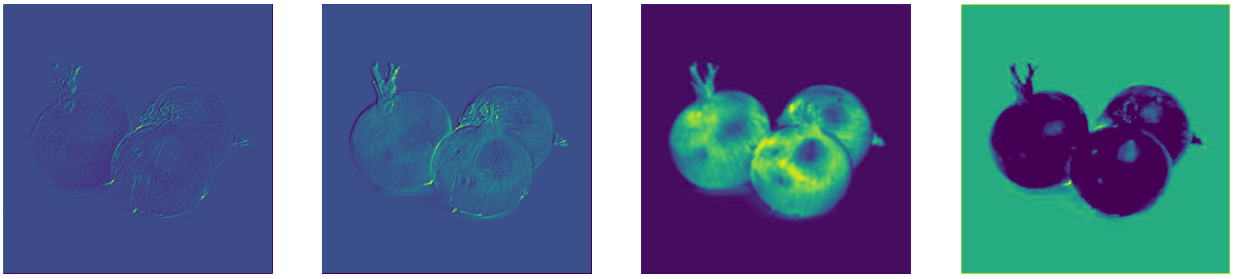
Fig. 2: CNN Architecture



(a) block1_conv1_1 (b) block1_conv1_2 (c) block1_conv1_3 (d) block1_conv1_4

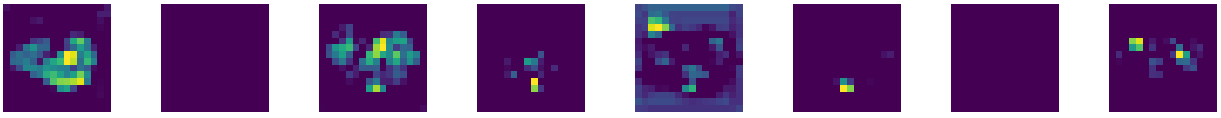


(e) block1_conv2_1 (f) block1_conv2_2 (g) block1_conv2_3 (h) block1_conv2_4

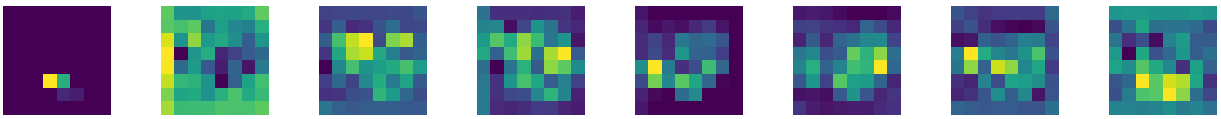


(i) block1_pool_1 (j) block1_pool_2 (k) block1_pool_3 (l) block1_pool_4

Fig. 3: Visualization of VGG block 01 that includes two convolutional layers each with 64 filters plus a max pool layer



(a) block5_1 (b) block5_2 (c) block5_3 (d) block5_4 (e) block5_5 (f) block5_6 (g) block5_7 (h) block5_8



(i) conv_1 (j) conv_2 (k) conv_3 (l) conv_4 (m) conv_5 (n) conv_6 (o) conv_7 (p) conv_8

Fig. 4: Visualization of VGG block 05 and extra convolutional layer followed by batch normalization, pooling and dropout

drop or not. We also observe a slower rate of converging when more dropouts in placed, but each epoch only took relatively less time to complete.

5) Dense Layers: Convolutional layers are the main suppliers of high-level features. Fully connected (i.e., Dense) layers combine different features to an actual neural network. The size and number of dense layers are always major

concerns in our experiments.

All together with VGG model, we have 14 convolutional layers (13 VGG layers plus one extra) to learn the sort of features for better predictions. First, we don't want to increase the number of parameters, but in the same time we want to fit the convolutional features to an actual fully connected network. As presented in Table II, we use three dense layers close to the output layer of 7 classes. First dense layer has 128 neurons, and 64 and 16 to follow in the next two layers. In the complete model, we only account for 14% of parameters to train. Dense layers equipped with 3% out of trainable parameters. The decision of having three connected layers with the given set of sizes is taken after a process of trial-and-error.

We achieve a greater detail of success with softmax activation function than ReLU in dense layers. Fully connected layer represents a linear relationships of convolutional features. With the softmax, it could be able to learn non-linear relationships in a multi-classification problem. Together with a stack of three dense layers, softmax found to be better with the accuracy of predictions.

Layer (type)	Output Shape	Parameters
InputLayer	(None, 512, 512, 3)	0
vgg16 (Model)	(None, 16, 16, 512)	14714688
Conv2D	(None, 16, 16, 512)	2359808
BatchNormalization	(None, 16, 16, 512)	2048
Activation	(None, 16, 16, 512)	0
MaxPooling2D	(None, 8, 8, 512)	0
Dropout	(None, 8, 8, 512)	0
Dense	(None, 8, 8, 128)	65664
BatchNormalization	(None, 8, 8, 128)	512
Dense	(None, 8, 8, 64)	8256
BatchNormalization	(None, 8, 8, 64)	256
Dense	(None, 8, 8, 16)	1040
BatchNormalization	(None, 8, 8, 16)	64
Flatten	(None, 1024)	0
Dense	(None, 7)	7175

TABLE II: Model summary: Trainable parameters = $2,443,383(14\%)$, Non-trainable parameters = $14,716,128(86\%)$

C. Architecture Choices

1) Loss function: Since we have a multi-classification problem, where each image only belongs to single class, we use multinomial cross-entropy loss as the optimization objective. This decision is biased only to deal with the classes in categorical format.

2) Optimizer: We settle with *stochastic gradient descent (SGD)* method as our optimization algorithm with the parameters of learning rate and momentum. In the first place, SGD is selected for faster and stable convergence and to avoid the cost of weight update over back propagation with a set of small batches. We set the size of batches to 32 examples after few sets of experiments. Learning rate is set to 0.0001 with a momentum value of 0.85. Momentum was used to walk towards the global optima, but not to stuck in a local minima. We keep the momentum value relatively larger to have faster convergence, while very small learning rate to have small gradient updates.

We also experiment with Adam, though we achieve relatively less training loss, this method has not been generalized well for validation data. We're also motivated by a recent comparison of SGD over adaptive optimizers including Adams [7].

3) Regularization: In another attempt to solve the problem of over-fitting, we apply L2 regularization in both convolutional and dense layers. We set the regularization term to 0.0001. We select L2 over L1 regularization since we have non-sparse matrices to represent images.

We stop the training once we don't achieve any improvement of the validation loss for 20 epochs. This method is also a form of regularization.

IV. Evaluation and Results

We use Keras framework for the implementation of proposed ConvNet. As presented in Section III, VGG16 was used as the initial pre-trained model. Dataset is divided into training and validation sets in the first place. Images are rescaled into a fixed size during training, but keep the original aspect ratio, which then follow few pre-processing steps (Section II). We keep the VGG weights frozen, which remains 14% weights to train in our model. The whole model was trained over 100 epochs with the architecture choices presented in Section III-C.

A. Accuracy

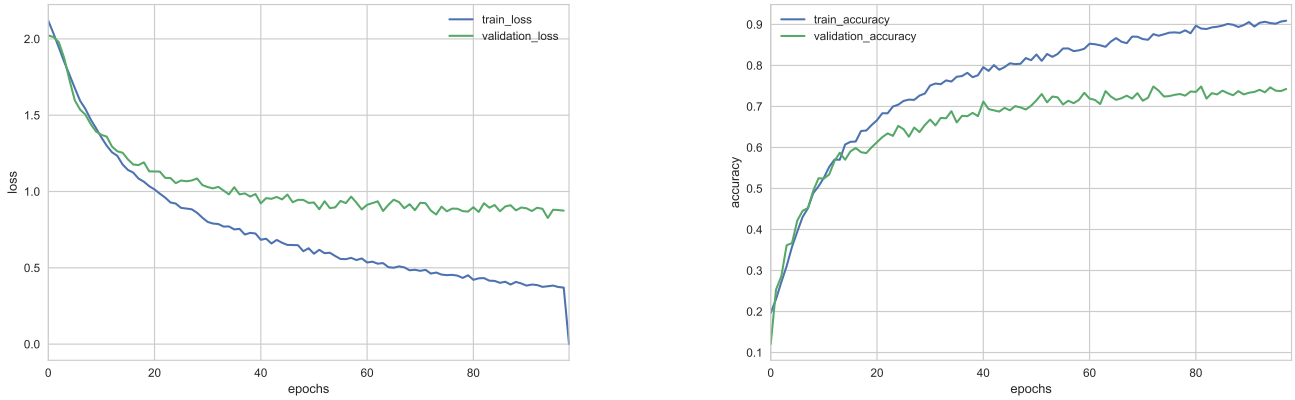
Figures 5a and 5b present the performance of trained ConvNet. We report metrics related to both training and validation datasets. Training accuracy reaches 90% after 100 epochs in placed. We had prior configurations where training accuracy reaches 100% rapidly, but that clearly shows over-fitting due to the poor performance over validation data.

Somebody could argue that the model could have more capacity to learn, since it doesn't reach the optimal loss value. But we're interested on the stable rate of learning new examples than achieved loss value after 100 epochs. The loss curve in training data reflects the model is learning better. The width of the curve represent the batch size, which might be too narrowed. But this is due to the selection of batch size.

Validation metrics present the goodness of model predictions over unseen data. Validation accuracy was better than training accuracy in the initial epochs. But we note the sudden decline of the model performance for unseen data after 20 epochs. The rate of learning new examples was superior than the rate of predicting unseen examples. But importantly, we observed a smaller gap between curves, which reflects less over-fitting.

In another note, early stopping was active in training, but didn't trigger the model to stop since the model is learning continuously.

Figure 6 presents the confusion matrix for predicted labels in validation data. The model seems to achieve better prediction results in some of the classes. As an example, the accuracy of predictions is relatively good for sliced and



(a) Loss

(b) Accuracy

Fig. 5: Loss and accuracy

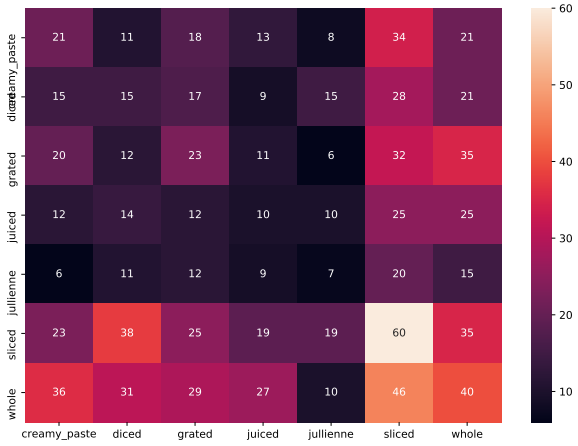


Fig. 6: Confusion Matrix for validation data

whole images, two classes that have relatively large fraction of training examples. However, the images with the state of whole have been predicted as many other classes. We will report the confusion matrix for test data as they release.

Here, we list our experience during the training and validation process.

1) The penalty of image augmentation: In our first try, we do all sort of image pre-processing including histogram equalization as presented in Section II. More recent experiments only consider feature centralization and normalization, which provides better accuracy with the softmax activation function applied over dense layers.

We resize the images to 128×128 in the initial set of experiments (i.e, crop the center), but gain higher success when the images are in the resolution of 512×512.

2) ConvNet Layers: Our results are better generalized with the given architecture. However we note that the accuracy is more dependent on the number of weights than the depth of layers. Specially, the size of dense layers effect the accuracy. But the number of extra convolutional layers effects less.

V. Discussion

In general ConvNet integrate two-fold methods of feature extraction and multi-class classification. Traditional feature extraction methods are unsupervised, such that it's hard to map feature spaces with the distinction of target classes. ConvNet learns features automatically in a supervised manner. Such that, feature spaces served as better clients to distinguish target classes.

VGG16 serves as the backbone in the proposed ConvNet model, which accounts 86% weight parameters. Since VGG16 was trained on a larger collection of images in ImageNet, we keep already trained weights frozen in the bottom convolutional layers. ImageNet is supposed to have many examples of cooking objects, but we don't exactly know the extent of variance exist in object states. Such that, VGG would give a set of under-estimated weights that trained on many other image classes out of our interest. This might be an issue when a model is not properly localized to the problem. But in the same time, ConvNet would perform well with a larger amount of training data. Since VGG uses small filters to stack together a set of convolutional layers, the extracted features would become more complicated and representative. But there are many hyper-parameters that need to be in an optimal combination for better predictions in a ConvNet architecture which are selected with a lot of trial-and-error experiments.

Our major concerns: *can we generalize the concepts of object state recognition through the proposed architecture?*. This is a major battle we would like to solve. But there are several limitations. Object states are causally related with human motions, which makes object state recognition in a widely open space of problems due to the significant

diversity of application domains. Such that we require a millions of examples to have better conclusions, specially with ConvNets. ConvNet could memorize training examples directly than embedded patterns [1]. To avoid this, we need a similar distribution of test data with a variety of object states to represent human motions.

REFERENCES

[1] Hossein Hosseini, Baicen Xiao, Mayoore Jaiswal, and Radha Poovendran. On the limitation of convolutional neural networks in recognizing negative images. In *Machine Learning and Applications (ICMLA), 2017 16th IEEE International Conference on*, pages 352–358. IEEE, 2017.

[2] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456, 2015.

[3] David Paulius, Yongqiang Huang, Roger Milton, William D Buchanan, Jeanine Sam, and Yu Sun. Functional object-oriented network for manipulation learning. In *Intelligent Robots and Systems (IROS), 2016 IEEE/RSJ International Conference on*, pages 2655–2662. IEEE, 2016.

[4] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[5] Yu Sun and Yun Lin. Modeling paired objects and their interaction. In *New Development in Robot Vision*, pages 73–87. Springer, 2015.

[6] Yu Sun, Shaogang Ren, and Yun Lin. Object–object interaction affordance learning. *Robotics and Autonomous Systems*, 62(4):487–496, 2014.

[7] Ashia C Wilson, Rebecca Roelofs, Mitchell Stern, Nati Srebro, and Benjamin Recht. The marginal value of adaptive gradient methods in machine learning. In *Advances in Neural Information Processing Systems*, pages 4151–4161, 2017.