Cooking Objects States Identification Model Training Trial

Zihe Ye

***Abstract*— Objects classification has become a hot topic in the past few years, and there are many tools to easily identify objects such as human, animals, cars, and others. In robotics cooking, a robot should be able to identify not only what the object is but also the state of the object to perform the next step. However, there is no very high accuracy model can solve this issue today. This paper talks about trials on building a model for cooking objects state classification using no pretrained model and got a 58% accuracy through different model settings, and it helps to better learn what factors affect such model.**

***Keywords*—Convolutional Neural Network, State classification, Learning rate, Overfitting, Hyperparameters.**

I. Introduction

As more deep learning models have been developed, robots can do many things to help reduce human labor involved in many tasks, including sweeping the floor, doing translations, and a lot more. It will be a good thing if robots can replace human to cook to save more time and to help handicapped people to do what they cannot do.

It is easy for a healthy person to do cooking, and we can simplify the jobs when cooking as learn recipe, recognize cooking objects, and perform correct actions on them at the right time, this will be a sequence with time limit. There are several difficulties involved for a robot to do correctly: recognizing correct recipe and understand the recipe [1-2], predicting coming cooking tasks [3-4], objects recognition and classification [5], grasping objects [6], and some other less important tasks. It is important to understand that robots cooking is not simply traditional image recognition and classification because robots must relate action with states, thus a concept called FOON (Functional object-oriented network) is introduced by Paulius, David, et al [7].

States recognition on cooking objects is a multitask deep learning problem because it needs to retract common elements from objects with a lot more different elements such as color, shape, and a lot more. There have been some models working on states recognition recently [9-10], in these two cited papers, there are 7 states that an object can fall into.

A more challenging task has 11 states, and the dataset is larger, a paper worked on states classification [8] states a fine-tuned model can reach more than 80% accuracy.

This paper will work on the same issue with same dataset of the work done by A. B. Jelodar [8] using no pretrained model or commonly used architects such as VGG [12], ImageNet [13], Resnet [11]. The model was designed and run under python 3.8 environment on Google Colab with GPU computing resource support. And the best accuracy through two weeks trial reached 58% on the 1543 valid set testing.

The following sections will be: Section2 talks about data and preprocessing, Section3 talks about Methodology, Section4 talks about Evaluation and Results, Section5 is the discussion section.

II. Data and Preprocessing

A. Data

The Dataset for this paper includes a training set of 7213 images and a valid set of 1543 images. These images can be divided into 11 classes as shown in figure1.

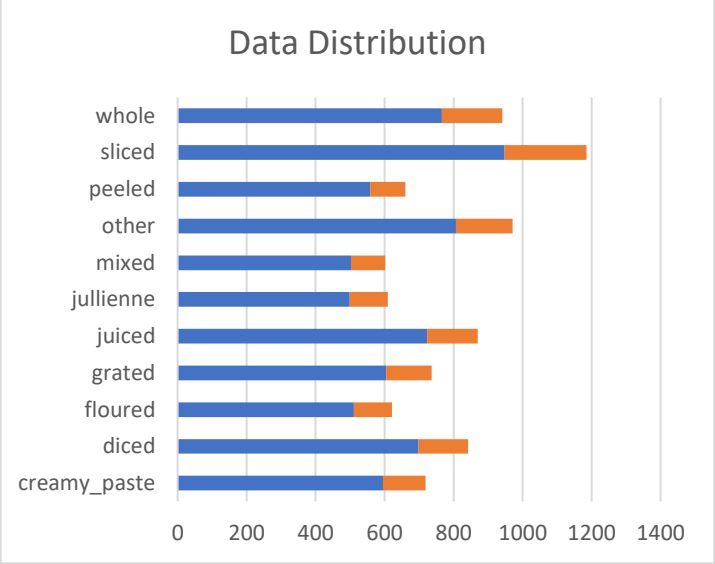


Figure1

The original images and classification were both done manually to ensure the accuracy. Each class of data is stored in a folder with the class name, and there are 11 class folders inside train folder and valid folder for later data loading step.

B. Data preprocessing

This subsection will talk about data loader, and class functions in the dataset.py file as well as data augmentations. There is a given args.py file, and some important hyper parameters for both dataset class and model are given all with default values, the arguments in the args.py file related to data preprocessing are: data_dir which is the root directory of where the train and valid are stored, the default value is "data"; mode, which is to indicate which mode data we want to get, by

default the mode is set to train to load trainset, and we can change the mode to valid and test too; image_size is the standard length of each side in an image, by default the value is 256, means an image is composed of 256*256 pixels; crop_size is for transformations to crop all images to crop_size*crop_size, and the value is by default 224*224, this is because 224 is what VGG only accepts; batch size is how many images to process for each round, by default the batch size is set to 128.

We have a StatesDataset class inherits from Dataset of data library. Inside the StatesDataset class, a batch of data is loaded each time called getloader. All other methods such as shuffle, get_item is override on the original super class implementation to make the training process correct and efficient.

We have two different transform lists for data augmentation, one is for training set including RandomCrop, RandomHorizontalFlip, RandomVerticalFlip, and ColorJitter with contrast parameter equal to 1. The valid set and test set have a constant method for data transformations including Resizing, and CenterCrop. Both transform lists transform the image to tensor and normalize at the end.

III. Methodology

This part is to introduce the steps building up the model, tuning methods and the main training and validation part.

The basic unit in PyTorch.nn is layer, and it is common to use Convolutional layer to retract features, pooling layers to maximize and summarize features, batch normalization layer to normalize the batch, activation layer to determine which feature to be sent to the next layer, and fully connected linear layer for final classification. All the layers mentioned above are used in the trial process and the final model this paper is based on. The following figure 2 is the final layout of the model submitted with this paper:

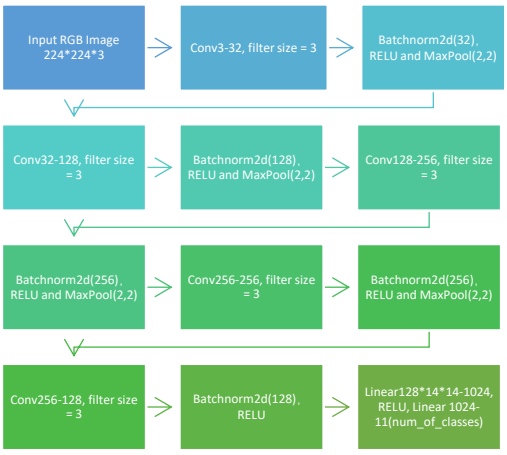


Figure 2

I have tried different combinations by changing the numbers of convolutional layers, the filter size, the channel combinations, with and without pooling or dropout layers, as well as the connection of fully connected layers on the model.

To make a model better fits the dataset to get higher accuracy and lower loss, I also tried different transforms and hyperparameters including learning rate, batch size, and crop size with image size. Some trials are better than others, and there is no certain rule to try different models, as the only purpose is to find a better model through trying and tuning different factors.

This section illustrates some of the high-level experience and lessons I got from changing the model, and the next Evaluation and Results section illustrates the loss and accuracy of different model settings in details.

My path is from easy to complicate. My first model is composed of three convolutional layers and a linear layer without other layers in between, the accuracy goes up in the first 10-20 epochs then reach the ceiling at around 25% and stopped growing. Then I added activation layers RELU and Pooling layers after each convolutional layer, and the accuracy reached 32%. More convolutional layers are added later to increase the accuracy, with all RELU and pooling after each convolutional layer resulted in generating overfitting starting at epoch 25. The first submitted version reached 52% at epoch which is comprised of 4 convolutional layers. The final model is with six convolutional layers and batch normalization, RELU, Max pool after each convolutional layer except for the last layer, following are two linear layers from what the convolutional layers left to 1024 to 11 (number of classes to classify), and resulted in a 58% accuracy reached at epoch 60 in the validation set.

IV. Evaluation and Results

A. *Evaluation*

This section illustrates in details different configurations I tried and is with a results analysis and summary. Figure 3 and Figure 4 represent accuracy curve and loss curve for the total 100 epochs respectively of the final model.

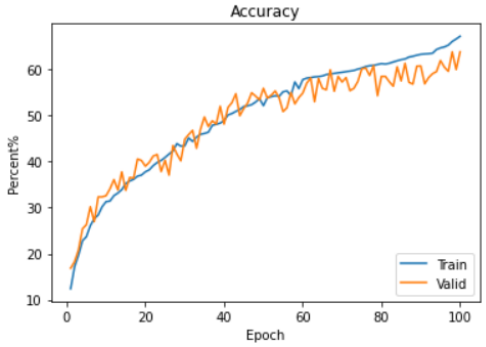


Figure 3

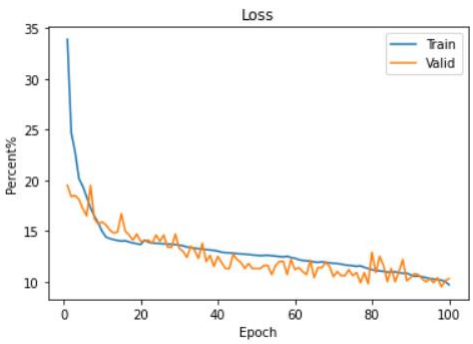


Figure 4

The changes I did can be divided into change the model, change the dataset, and change hyperparameters.

The following table 1 is the change of model and results, and table 2 is the result of change on dataset (data augmentation), the table 3 is the result of change on hyperparameters.

Model Settings	Training Accuracy/Loss	Validation Accuracy/Loss	#Epoch
3 Conv	0.26/0.18	0.28/0.18	50
3 Conv w Pool	0.34/0.16	0.36/0.16	50
5 Conv w Pool	0.41/0.14	0.43/0.14	60
7 Conv wo Pool	0.33/0.17	0.32/0.17	50
w. Dropout	No big change	No big change	

Table 1

The above table 1 is under the condition without activation function and batch normalization, and the experiment shows activation function after every pack of layer can increase accuracy and lower loss while batch normalization helps to deal with overfitting.

Data Augmentation	Training Accuracy/Loss	Validation Accuracy/Loss	#Epoch
Random Crop	0.61/0.12	0.58/0.12	70
Center Crop	0.56/0.13	0.54/0.13	70
wo. RandomHorizontal Flip	0.52/0.13	0.52/0.13	60
wo. RandomVertical Flip	0.53/0.13	0.52/0.13	60
wo. ColorJitter	0.59/0.12	0.58/0.12	70

Table 2

The above table shows the affect of data augmentation options, it is clear that random flip has large positive affect on the accuracy and loss, while ColorJitter also has some affect but not as obvious.

Hyper Parameters	Training Accuracy/Loss	Validation Accuracy/Loss	#Epoch
Batch Size=64(default 128)	0.57/0.12	0.57/0.12	60
Learning rate=0.0001	0.48/0.14	0.49/0.14	90
Learning rate=0.01	0.53/0.13	0.52/0.14	25

Table 3

The Table3 shows that batch size has little affect on the training and validation accuracy, while learning rate plays a more important role in the training of this model. However, there could be more things to play around with learning rate including gradual deduction of learning rate in later epochs, and change optimizers, etc.

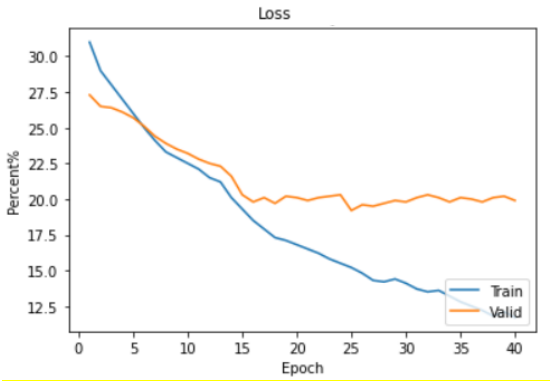
B. Results

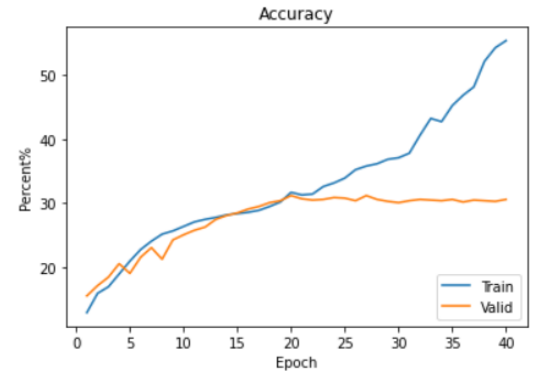
After trying different combinations, I found the current model can reach to 58% accuracy at epoch 60 and stay relatively stable afterwards before the last epoch. And several known helpful points I notices include: 1. Data augmentation is important for training especially when dataset is small. 2. More types of layers can help increase the complicity of the model at the beginning and help increase the accuracy faster. 3. If the model is too complicated and starts to show overfitting, then apply batch normalization and dropout can significantly help reduce the overfitting phenomena. As the pages are limited, not every factor is known or listed, but it is realized that channel numbers, filter size also play important role when training the mode.

The final model setting is shown in the section 2, and the data augmentation is introduced in the data preprocessing part. The final accuracy is 58%-60% in the last 30-40 epochs and relatively stable without significant signal of overfitting, so this should be considered as a good starting model for later tuning.

V. DISCUSSION

After finishing this project, I found somethings clear and a lot of others unclear and to be explored in the future. I learned that the regular path to efficiently build a model able to use is to stack up layers at the beginning until overfitting, then we can play around with tools such as dropout layers, reduce learning rate, change optimizer, etc, and all these knowledges can be gained from other people's experience as well as trying myself. An interesting thing I want to talk about in this section is the experience with overfitting when doing trials. The overfitting's reason was because of very complicated model but because of data processing. What I experienced was it was very hard for me to reach a higher accuracy after got to 55%, so I opened the images in the train and valid set, and I found a lot of images are much larger than 224*224, so I directly went to change the image size and crop size hyper parameters and do padding in the transforms section. However, it turned out my model overfitted a lot, and the training accuracy keeps going up while validation accuracy stays around 32% after the epoch 15. The accuracy/loss curves of the first 30 epochs are shown below.





After searching and rethinking, I found that simply padding and crop to a unified size cannot solve the issue that input images size not equal issue because this will result in leaving a lot of same black pixels if an image was small. So, resizing can solve the issue in some extent and not result in errors in programs while padding and cropping requires more attention on if there could be an image causing problem. However, resizing is not the best way for image preprocessing, and researchers are still to find a better way to make different size input images to fit a way to be fed to the model without losing much information and features. K. He, et al in [14] mentioned a new way to deal with different input image size.

ACKNOWLEDGMENT

I want to show my gratitude to Dr. Sun for great lectures and Ahmad for being the TA of this class and provide us lots of great suggestions and help!

REFERENCES

[1] Meyers, A., Johnston, N., Rathod, V., Korattikara, A., Gorban, A., Silberman, N., Guadarrama, S., Papandreou, G., Huang, J. and Murphy, K.P., 2015. Im2Calories: towards an automated mobile vision food diary. In Proceedings of the IEEE International Conference on Computer Vision (pp. 1233-1241).

[2] Malmaud, J., Huang, J., Rathod, V., Johnston, N., Rabinovich, A. and Murphy, K., 2015. What is cookin'? interpreting cooking videos using text, speech and vision. arXiv preprint arXiv:1503.01558.

[3] Lade, P., Krishnan, N.C. and Panchanathan, S., 2010, December. Task prediction in cooking activities using hierarchical state space markov chain and object-based task grouping. In Multimedia (ISM), 2010 IEEE International Symposium on (pp. 284-289). IEEE.

[4] Liu, C., Cao, Y., Luo, Y., Chen, G., Vokkarane, V. and Ma, Y., 2016, May. Deepfood: Deep learning-based food image recognition for computer-aided dietary assessment. In International Conference on Smart Homes and Health Telematics (pp. 37-48). Springer, Cham.

[5] R. Arbiol, V. Pala, Institute Cartogràfic de Catalunya (ICC), Parc de Montjuïc, E-08038 Barcelona, Spain, Y. Zhang, University of New Brunswick, Department of Geodesy and Geomatics engineering, P.O. Box 4400, Fredereicton NB, E3B 5A3, Canada, ADVANCED CLASSIFICATION TECHNIQUES: A REVIEW.

[6] Sun, Yu, and Yun Lin. "Systems and methods for planning a robot grasp that can withstand task disturbances." U.S. Patent No. 9,649,764. 16 May 2017.

[7] Paulius, David, et al. "Functional object-oriented network for manipulation learning." Intelligent Robots and Systems (IROS), 2016 IEEE/RSJ International Conference on. IEEE, 2016.

[8] A. B. Jelodar, M. S. Salekin, Y. Sun. Identifying Object States in Cooking-Related Images. arXiv preprint arXiv: 1805.06956, May 2018.

[9] R. Paul. Classifying Cooking Object's State using a Tuned VGG Convolutional Neural Network. arXiv preprint arXiv: 1805.09391, May 2018.

[10] A. Sharma. State Classification with CNN. arXiv preprint arXiv: 1806.03973, June 2018.

[11] K. He, X. Zhang, S. Ren, J. Sun. Deep Residual Learning for Image Recognition. arXiv preprint arXiv: 1512.03385, Dec 2015.

[12] K. Simonyan, A. Zisserman. Very Deep Convolutional Networks for Large-Scale Imgae Recognition. arXiv preprint arXiv: 1512.03385, Sep 2014.

[13] J. Deng, W. Dong, R. Socher, L. Li, Kai Li and Li Fei-Fei, "ImageNet: A large-scale hierarchical image database," 2009 IEEE Conference on Computer Vision and Pattern Recognition, Miami, FL, USA, 2009, pp. 248-255, doi: 10.1109/CVPR.2009.5206848.

[14] K. He, X. Zhang, S. Ren, and J. Sun, "Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition," in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 37, no. 9, pp. 1904-1916, 1 Sept. 2015, doi: 10.1109/TPAMI.2015.2389824.