The State Classification Challenge

Sudheer Nadella
Computer science
University of South Florida
Tampa, US
snadella@usf.edu

***Abstract*— Robotic cook needs to know the objects and its states for any cooking activities. There has been a lot of research carried out on object detection but object's state detection has not been studied well. Understanding object states is as important as object recognition for a robotic cook. In this paper, a custom deep learning model is introduced which addresses the state identification problem by identifying states from the images. The model was trained on 11 cooking states of 17 cooking objects and is analysed with different optimizers and batch sizes. The proposed model can be used to identify the states of the cooking objects to solve the state identification problem.**

I. INTRODUCTION

In the last decade, there has been a significant improvement in the usage of robots for several activities. Robots are playing a great role in many industries, such as manufacturing parts in the automobile industry, used for surveillance in Defense, and assisting doctors in performing surgeries. One of the interesting areas is how robots can be used in the kitchen for cooking. A robotic cook is very helpful for seniors and people with severe physical disabilities. For this, a robot should have knowledge of the objects/items used and their states along with the cooking instructions. For example, a whole vegetable has many states, such as whole, sliced, and diced. For making some item out of that vegetable, the whole vegetable has to be sliced and then diced. Given the cooking instructions, a robotic cook has to pick the vegetable, wash it, slice it and then dice it. If provided with an instruction that only requires sliced vegetables, and the robot was provided whole, sliced and diced vegetables, it has to pick the sliced ones and use them for cooking. So, understanding the states of the objects/items is as important as knowing the object/states. There are many significant works for object detection[1] but unfortunately, there are not many on object state detection.

In the kitchen environment robots needs to perform different tasks to achieve outcome of the planned tasks[2]. A robotic cook requires different types of grasping for grabbing different states of the object. For example, consider a potato, a whole potato is grabbed/picked in a different way than sliced or diced potato[3]. For slicing a potato it has to hold the whole potato in a different way. So, based on the objects and their states, it will decide on what type of grasp/operation is required[4] to perform assigned tasks.

In this paper a custom deep learning model[5], [6], [11] using convolution layers is proposed to identify the object states in the images. The data set contains 17 cooking objects with their 11 different states. A Convolutional Neural Network was built and trained on the data set to identify the states of the objects. Evaluation was done by comparing the validation performance of different optimizers and batch sizes. The rest of the paper is organized as follows. Section II discusses about data and preprocessing, section III discusses about the methodology. Section IV analyses and evaluates the results and further discussion is provided in section V.

II. DATA AND PREPROCESSING

A. Data set

The data set[7] contains 17 cooking objects with 11 different states[data]. It has total 7213 samples for training and 1543 samples for validation. Both training and validation data has 11 cooking states in them. They are creamy paste(719), diced(839), floured(621), grated(736), juiced(870), julienne(609), mixed(602), other(971), peeled(660), sliced(1185), and whole(941).

B. Data preprocessing

I have used ImageDataGenerator[8] for real-time data augmentation. Keras ImageDataGenerator transforms each image in the input batch of images passed to it by translations, rotations, re-scaling, etc. This process is also called "in-place" or "online" data augmentation because this augmentation is done at training time which returns only the randomly transformed data. Since the data that we have is not quite enough for a deep network, augmentation plays a great role and helps the network to be trained in a more generalized way. I have done Sample-wise centering[9] which changes the image mean to 0 by subtracting the mean value of the pixels from each pixel and Sample-wise normalization which divides the image by its standard deviation to normalize it. We should normalize the data to make sure each pixel has similar data distribution. I used a rotation range of 40-degrees, width and height shift of 20%, shear and zoom range of 20%, and used horizontal flipping. Augmentation methods and their settings has been shown in figure 1. I only used 20% for width shift, height shift, shear, and zoom range in order to maintain the maximum features while augmenting the images.

Method	Setting
Rescale	1/255
Rotation range	40
Width Shift	0.2
Height Shift	0.2
Shear range	0.2
Zoom range	0.2
Horizontal flip	True
Fill mode	Nearest

Fig. 1: Augmentation methods

III. Methodology

A custom deep learning model built with multiple convolution layers[10] is proposed to identify the states of the cooking objects. Before CNN became popular, people used to manually create features from images and feed those features to some classification algorithm like SVM[12]. Where as convolution layers[13] effectively uses the adjacent pixel values to extract features and down sample the image and there is no need of manual feature extraction. Convolution preserves the spatial relationship between pixels by learning image features using small squares of input data. Since convolution neural networks(CNNs) automatically detects the important features without any human supervision, they are mostly used for image recognition and classifications. The main reason why I am using convolution layers in this model is because of its efficient feature extraction and its performance on image data. The proposed model was trained on the data that is augmented using ImageDataGenerator as discussed in the previous section. I have used activation functions Relu[14] in between the layers and Sigmoid at the end. An activation function is nothing but a node that decides what is to be passed to the next neuron. The values are calculated based on the activation functions selected.Relu passes the value which is maximum of 0 and the value itself, that means it passes the same value to the next neuron if the value is positive but passes 0 if the value is negative. I have also used Batch Normalization[15], [19], a technique to standardize the inputs to a network. Batch Normalization yields faster training and higher accuracy. I have applied Batch Normalization after activation function to normalize the outputs of the activation layer. Pooling layer[16], [20] usually reduces the features size. The most common pooling techniques are Min Pooling, Avg Pooling, and Max pooling. Pooling layer frame/window size as input and extract features from that frame/window. In this model, I have only used Max pooling, which takes the maximum value in that frame and discards the other values. Dropouts are used to prevent the model from over fitting. I have used two dropouts[17], [18], on with 0.15 and the other with 0.5. The concept of dropout is in every run the given percentage of neurons will be dropped and the data has to be learned by the remaining neurons. This makes the model more generalized, as every other neuron will be doing the dropped neuron learning which increases generalization and improves the overall performance. Flatten layer will flatten all the data coming from the above layers to pass it to the Dense layers. I have used 2 dense layers, one with 64 neurons and the other with 11 neurons(for output). The final Dense layer has 11 neurons since we have 11 classes and each of the 11 neurons will give the probability of the image belonging to that class. I used callbacks to save the best model. The proposed model has 6 blocks, dropouts, flatten layer and two dense layers with activation functions. Each block has a convolution layer, activation function, Batch Normalization, and Max pooling except there is no Max pooling in the 5th block. The network of the proposed model is illustrated in figure 2. I have tried using various batch sizes (16, 32, and 64) and optimizers(SGD, RMSprop and Nadam) and Nadam with batch size 64 is found so far the best with our model and dataset.

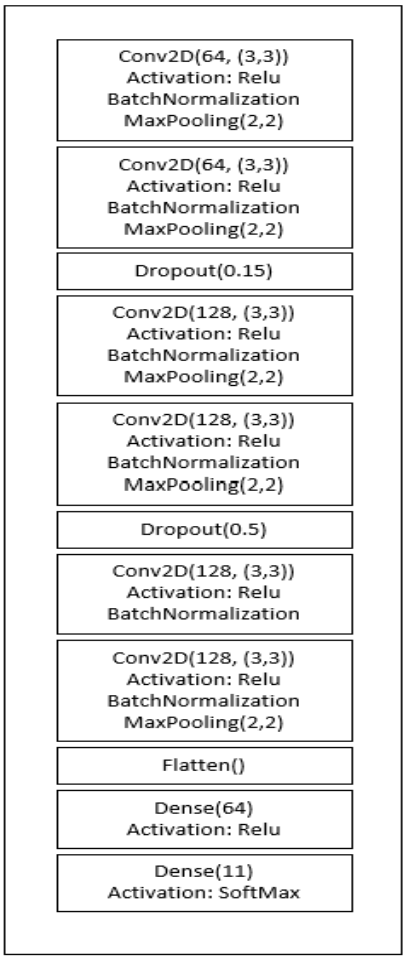
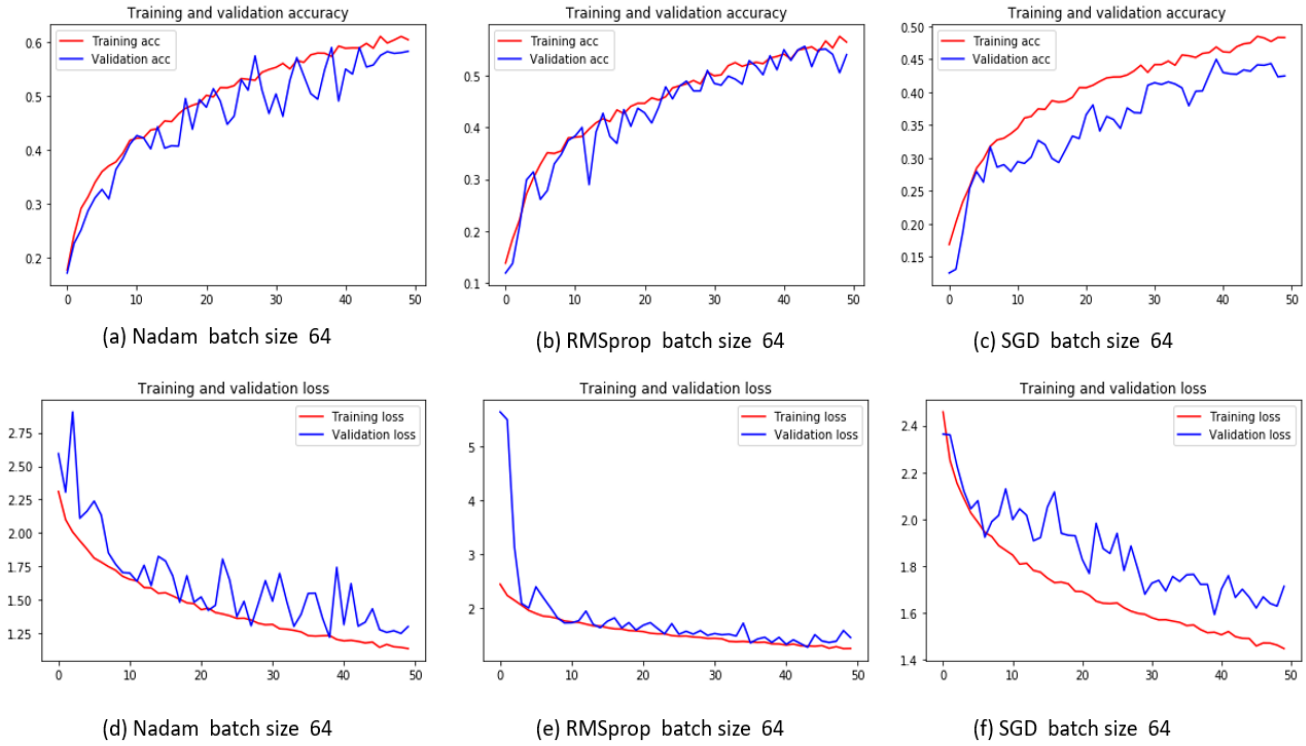


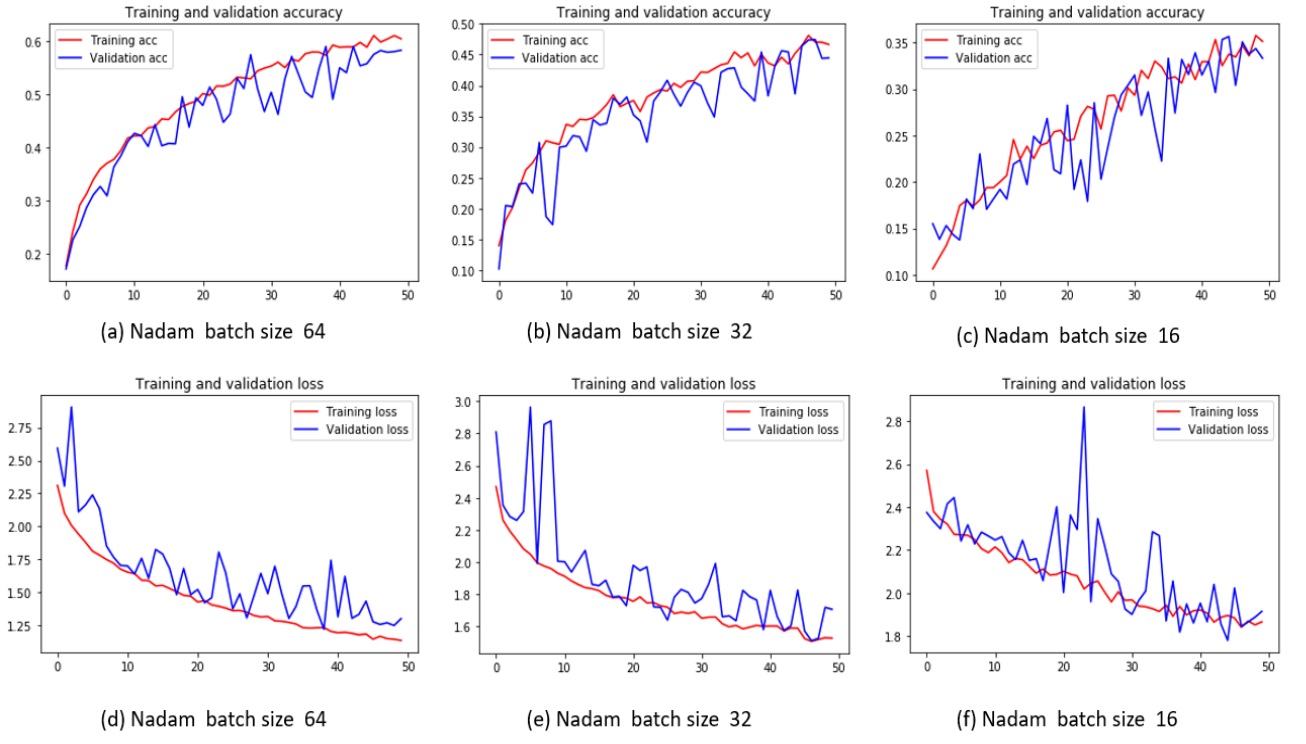
Fig. 2: Proposed model

IV. Evaluation and results

The proposed model was evaluated based on different parameters. I have tried to analyse the result in terms of different batch size and optimization techniques. Each of the experiments/run is done at 50 epochs. The model was selected based on the maximum validation accuracy and minimum validation loss.



(a) Validation accuracy and loss during training for different optimizers



(b) Validation accuracy and loss during training for different optimizers

Fig. 3: Training and validation accuracy under different conditions

A. Based on optimizers

For this project, I have tried using Nadam, RMSprop, and SGD to find out the best optimizer for the model. Experimentally, I found Nadam and RMSprop are performing better and SGD did not show good results. As you can see in figure 3(a), Nadam and RMSprop showed similar pattern where SGD did not perform up to the mark. Out of Nadam and RMSprop, Nadam validation accuracy was around 57% and validation loss at 1.4 where RMSprop validation accuracy was around 53% and validation loss at 1.7. So, Nadam performed best among the three. I have tried many learning rates but found the best learning rate for the model as 0.01. All the experiments are carried out for 50 epochs. RMSprop has also shown good results but not the best when compared with Nadam. Might give better accuracy than this at higher epochs.

B. Based on batch size

On running several experiments, I observed the best results are obtained at batch size 64. For this model trained on the provided dataset, 64 is the optimal batch size. I have compared the validation performance using batch sizes 16, 32, and 64. You can see how the performance has varied for different batch sizes in figure 3(b). Looking at the validation accuracy and validation loss in the figure 3(b), we can say that batch size 64 shows the best results when compared to batch size 16 and 32 in terms of validation accuracy and loss.

V. Discussion

The proposed model was able to achieve a classification accuracy of 60% in the experiments carried out with Nadam as optimizer using a batch size of 64. Better accuracy could be achieved by using the pre-trained models and training the model on more data. If we could have had the final test set the model will be evaluated on, we can clearly see for which class(state) the model is performing well and for which class(state) it is performing poor by plotting the confusion matrix. Generally if the model has poor performance on a particular class, that means the class consists of variety of items.

References

[1] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In CVPR, pages 770–778. IEEE Computer Society, 2016.

[2] Sun, Yu, and Joe Falco. "Robotic Grasping and Manipulation: First Robotic Grasping and Manipulation Challenge, RGMC 2016, Held in Conjunction with IROS 2016, Daejeon." (2018).

[3] Sun, Y. et al. "Robotic grasping for instrument manipulations." 2016 13th International Conference on Ubiquitous Robots and Ambient Intelligence (URAI) (2016): 302-304.

[4] Lin, Y., Sun, Y. (2014) Grasp Planning Based on Grasp Strategy Extraction from Demonstration, IROS, pp. 4458-4463.

[5] Jelodar, Ahmad Babaeian, Md Sirajus Salekin, and Yu Sun. "Identifying object states in cooking-related images." arXiv preprint arXiv:1805.06956 (2018).

[6] Salekin, Md Sirajus, Ahmad Babaeian Jelodar, and Rafsanjany Kushol. "Cooking state recognition from images using inception architecture." 2019 International Conference on Robotics, Electrical and Signal Processing Techniques (ICREST). IEEE, 2019.

[7] https://arxiv.org/abs/1805.06956

[8] https://keras.io/api/preprocessing/image/

[9] https://keras.io/api/preprocessing/image/

[10] https://keras.io/api/layers/convolutionlayers/

[11] Jelodar, Ahmad Babaeian, and Yu Sun. "Joint Object and State Recognition Using Language Knowledge." 2019 IEEE International Conference on Image Processing (ICIP). IEEE, 2019.

[12] Xiaowu Sun, Lizhen Liu, Hanshi Wang, Wei Song and Jingli Lu, "Image classification via support vector machine," 2015 4th International Conference on Computer Science and Network Technology (ICCSNT), Harbin, China, 2015, pp. 485-489, doi: 10.1109/ICCSNT.2015.7490795.

[13] M. Jogin, Mohana, M. S. Madhulika, G. D. Divya, R. K. Meghana and S. Apoorva, "Feature Extraction using Convolution Neural Networks (CNN) and Deep Learning," 2018 3rd IEEE International Conference on Recent Trends in Electronics, Information Communication Technology (RTEICT), Bangalore, India, 2018, pp. 2319-2323, doi: 10.1109/RTEICT42901.2018.9012507..

[14] https://keras.io/api/layers/activations/

[15] https://keras.io/api/layers/normalizationlayers/batchnormalization/

[16] https://keras.io/api/layers/poolinglayers/

[17] https://keras.io/api/layers/regularizationlayers/dropout/

[18] Srivastava, Nitish et al. "Dropout: a simple way to prevent neural networks from overfitting." J. Mach. Learn. Res. 15 (2014): 1929-1958.

[19] Ioffe, Sergey, and Christian Szegedy. "Batch normalization: Accelerating deep network training by reducing internal covariate shift." International conference on machine learning. PMLR, 2015.

[20] Wu H., Gu X. (2015) Max-Pooling Dropout for Regularization of Convolutional Neural Networks. In: Arik S., Huang T., Lai W., Liu Q. (eds) Neural Information Processing. ICONIP 2015. Lecture Notes in Computer Science, vol 9489. Springer, Cham. https://doi.org/10.1007/978-3-319-26532-26