# Recognizing Cooking Ingredient States With A Fine Tuned VGG Neural Network

Shanice Clarke
Department of Computer Science and Engineering
University of South Florida
Tampa, FL, USA

***Abstract*— The task of object recognition is a widely researched problem in the field of deep learning. For some applications, such as cooking, recognizing the state of an object is just as important but this problem has not been discussed as much. For robotics research, the state of an object is important because it affects the way a robot must grasp and handle the object. The purpose of this paper is to present a convolutional neural network (CNN) that is used to recognize 11 different states of cooking ingredients in images. The neural network is given images of different ingredients such as an image of a diced onion and the network should classify it as diced. A VGG16 convolutional neural network was fine tuned with a dataset of 17 different cooking ingredients and fine tuned to achieve an accuracy of 76% on average.**

***Index Terms*— Convolutional Neural Network, Object State Classification, Transfer Learning**

## I. INTRODUCTION

Image classification is a popular subject in deep learning and artificial intelligence. Humans can easily and quickly recognize objects and their features. For a computer or robot to learn this seemingly simple task, it takes an enormous amount of examples for the artificially intelligent entity to be able to achieve this with an accuracy that is human-like. This project focuses on image classification as it relates to the various states of cooking items in images. The subject of object state classification may not be very popular but during a cooking activity, food objects change states frequently as they are being prepared. For tasks such as analyzing a cooking video to study a recipe, each state that an ingredient goes through needs be recognized so that the manipulations required to transition from state to state can be learned.

The rest of the paper is organized as follows: section II discusses background information on the subject of object state recognition, section III discusses the methodology of this work, section IV is an evaluation and results of the experiments and section V concludes this paper.

## II. BACKGROUND

### A. State Identification Challenge

The state identification challenged was introduced in [7] and defines the state of an object as the various physical shapes that the object can be transformed into by the manipulation of a human or robot. Identifying the various states of an object is important for tasks where a robot must manipulate an object. The robot would need to recognize the beginning state of the object as well as the state of the object during and after manipulation to be able to determine if the desired outcome has been achieved. For example, if a robot is given a whole onion that is not peeled it should be able to recognize that so that it would first peel the onion, then proceed to slice the onion and then dice. The steps to achieve the dice would be different depending on the beginning state of the onion.

Recognizing the state of objects is especially important for robotic grasping tasks. A robot would need to grasp a whole onion differently than how it would grasp a half of an onion or one that is diced. The robot would also need to know the state of the ingredient to determine what type of cooking utensil is needed to interact with it in its current state.

### B. Dataset

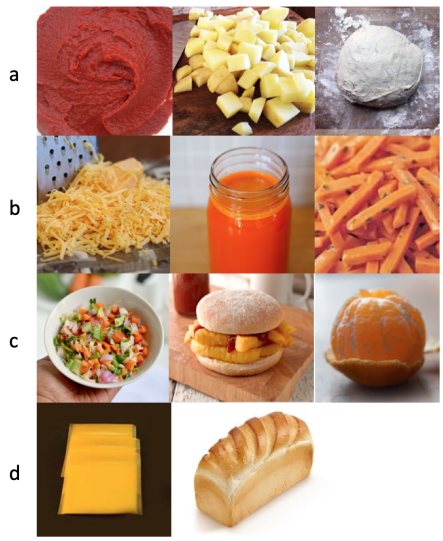


Fig. 1: Images representing the 11 states from the dataset a. creamy/paste, diced, floured b. grated, juiced, julienne c. mixed, other, peeled d. sliced, whole

To tackle the state identification challenge, a dataset [6] was collected for by Jelodar et al. [7] using the Google search engine and labeled manually and with the labelbox tool [1]. The dataset used contains 9,309 images of 17

cooking ingredients (beef/pork, bread, butter, carrot, cheese, chicken/turkey, dough batter, egg, garlic, milk, mushroom, onion, other, pepper, potato, strawberry, tomato). There are 11 different states for the ingredients (creamy/paste, diced, floured, grated, juiced, julienne, mixed, other, peeled, sliced and whole). The ingredients and states labeled as other include any ingredient or state not listed. Images for the training set were made up of 6,348 images, the validation set was 1,376 of the images and the test set was made up of 680 images.

## III. METHODOLOGY

### A. Image Preprocessing

To prepare the images for the neural network several steps were taken. The images were first resized to (224, 224) to match the input size for the VGG network. They also had to have the three RGB channels so that the network input is (224, 224, 3). The pixel values in the images were scaled to the range [0, 1]. This was done to help with convergence [9]. Data augmentation was also done to increase the number of images used for training. The images were flipped horizontally and rotated with a rotation range of 25. Zoom and shear options were also used in Keras [2]. Zoom was used with a range of [.8, 1.2] and shear was used with a shear intensity of 0.2. Figure 2 below shows an example of some of the augmentation that was done to the images.

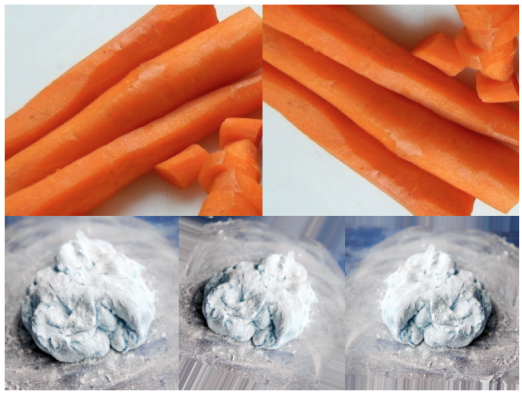

Fig. 2: Examples of images before and after augmentation

### B. Base Model

In deep learning, transfer learning is a method where a model that was developed for on task is reused to help the learning of another task. What has been learned in one setting is exploited to imporve generalization in another setting [4]. This is especially helpful for tasks where the amount of data for training is limited. It also helps to reduce the computational resources and time needed to train the new model. This project uses transfer learning by utilizing the VGG16 deep neural network which was pre-trained on the Imagenet [3] [10] dataset.

The layers and weights of the base model were used until after the fourth max pooling layer. These layers were frozen so they wouldn't be trainable with the rest of the network.

### C. Fine Tuning

Table 1 shows the structure of the network used for this project. Three convolutional layers were added to the network. After the first two convolutions, a batch normalization layer, max pooling and dropout layer followed them. The batch normalization was added to normalize the data being passed into the next layer. And the dropout was used to help prevent overfitting of the model to the data. Batch normalization, average pooling and a dropout layer were added after the next convolutional layer. Average pooling was chosen because the use of average pooling encourages the network to identify the complete extent of the object [11]. Both of the dropout layers so far have had a dropout rate of 0.25. All of the activation functions used are ReLU because of its ability to solve non-linear problems.

After flattening the dimensions of the network, a fully connected layer is used which is then followed by another batch normalization, dropout with a rate of 0.5 and then the softmax layer. The original softmax layer of the base model was used to classify 1,000 object classes so this layer was replaced with a softmax layer to classify the 11 food states to meet the needs of this project. The network output is the certainties of the image belonging to each of the 11 states. The state with the highest certainty is chosen as the classification. The loss function used was categorical cross entropy because this is a multi-class classification problem.

TABLE I: Fine Tuned VGG16 CNN

| Layers | |
|---|---|
| Input (224, 224, 3) | |
| Conv2D (224, 224, 64)<br>Conv2D (224, 224, 64) | |
| Max Pooling | |
| Conv2D (112, 112, 128)<br>Conv2D (112, 112, 128) | |
| Max Pooling | |
| Conv2D (56, 56, 256)<br>Conv2D (56, 56, 256)<br>Conv2D (56, 56, 256) | |
| Max Pooling | |
| Conv2D (28, 28, 512)<br>Conv2D (28, 28, 512)<br>Conv2D (28, 28, 512) | |
| Max Pooling | The base model was used until this layer. |
| Conv2D (14, 14, 512)<br>Conv2D (14, 14, 512) | The following layers were added to the network |
| Batch Normalization<br>Max Pooling<br>Dropout 0.25 | |
| Conv2D (7, 7, 512) | |
| Batch Normalization<br>Average Pooling<br>Dropout 0.25 | |
| Flatten<br>Fully Connected 1024 | |
| Batch Normalization<br>Dropout 0.5 | |
| Softmax 11 | |

### D. Evaluation and Results

Experiments were conducted to test the effects of different optimizers. The optimizers that were studied were Adam and RMSProp. RMSProp [5] is an adaptive learning rate optimizer that divides the learning rate of a weight by the average magnitudes of recent gradients for that weight [8]. The RMSProp optimizer was used with the following parameters:

- learning rate: .001, .0001
- $\rho$: 0.9
- $\varepsilon$: $10^{-7}$

The Adaptive Moment Estimation (Adam) optimizer [8] is a gradient descent algorithm that keeps an average of the squared gradients for each weight. The Adam optimizer was used with the following parameters:

- learning rate: .001, .0001
- $\beta_1$: 0.9
- $\beta_2$: 0.999
- $\varepsilon$: $10^{-7}$

Both of these optimizers were chosen because of their ability to update the learning rate which would help with convergence. Both of the optimizers were used with learning rates of .001 and .0001 but only the results of the learning rate of .001 are presented in this paper. Each of these optimizers were used with and without learning rate decay. The learning rate decay was set to the learning rate divided by the number of epochs of 100. Figure 3 and Figure 4 show the training and validation loss through the training process for both of the optimizers. Figure 6 shows the training and validation accuracy for each optimizer without learning rate decay through the training process.

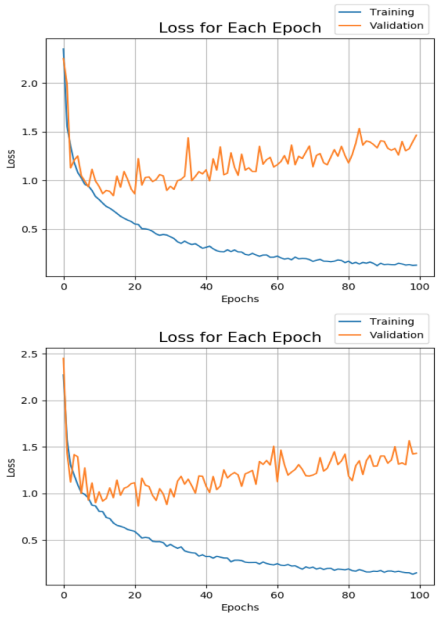


Fig. 4: Proposed Model with RMSprop Optimizer
Top: Training and validation loss with learning rate decay
Bottom: Training and validation loss without decay

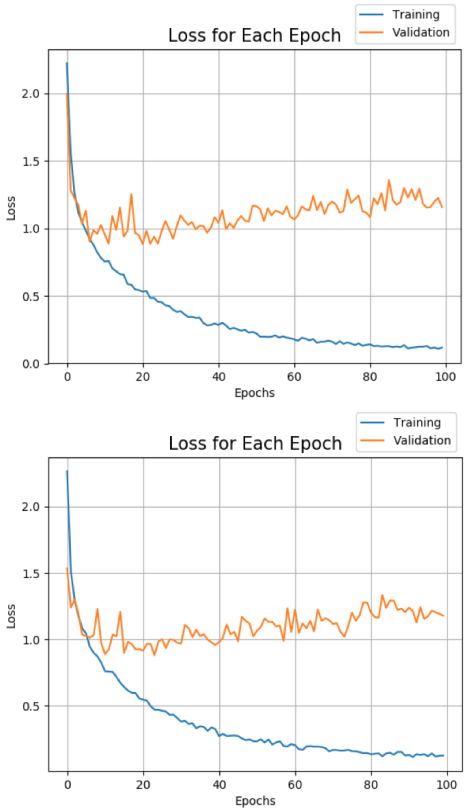


Fig. 3: Proposed Model with Adam Optimizer
Top: Training and validation loss with learning rate decay
Bottom: Training and validation loss without decay

During testing on unseen data, the Adam optimizer achieved an accuracy of 75.44% with learing rate decay and an accuracy of 76.03% without it. The RMSProp optimizer had a slightly lower accuracy of 71.18% with decay and 73.09% without it. Figure 5 shows examples of misclassified images. Some cooking states such as julienne and sliced or peeled and whole can look similar in certain images and therefore the model will be more likely to misclassify them. Also some classes have less images so there aren't as many training examples for the network to learn these classes as much as some of the other classes.


Fig. 5: Misclassified Images
a: Julienne classified as sliced
b: Sliced classified as julienne

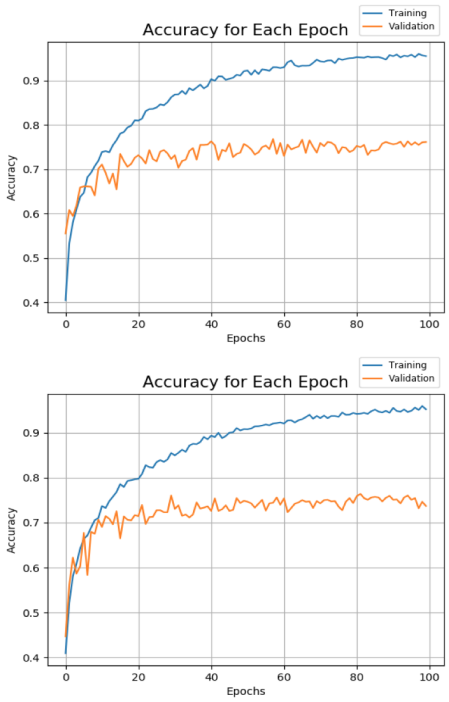


Fig. 6: Accuracy during training without learning rate decay
Top: Training and validation accuracy with Adam optimizer
Bottom: Training and validation accuracy with RMSProp optimizer

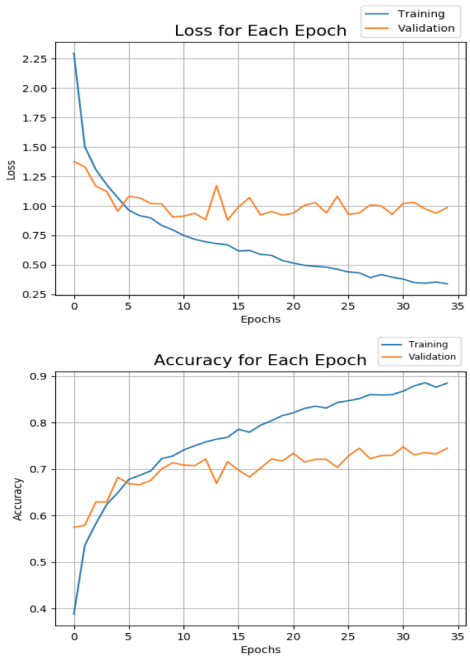


Fig. 7: Training using Adam Optimizer with early stopping
Top: Training and validation loss
Bottom: Training and validation accuracy

### *E. Overfitting*

The model had a tendency to overfit after the first 20 epochs so early stopping was added to the model with the Adam optimizer without learning rate dacay. This experiment achieved an accuracy of 72.5% on the test set. Training stops if the validation loss doesn't improve after a patience of 25 epochs. After the early stoppage, the weights that gave the lowest validation loss are restored. The results of this experiment are shown in Figure 7.

A new model, shown in Table 2, was explored to combat the problem of overfitting. The same layers from the base model were used as in the original model in Table 1. This time, only one convolutional layer was added and then it was followed by three fully connected layers with batch normalization, and dropout of 0.5 after each of them. The dropout factor was increased from 0.25 to 0.5. The idea here was that the base model already contained enough convolutional layers to sufficiently detect the image features so only one was added. The additional fully connected layers were added to help with classification. While there was a lot less overfitting, the accuracy was only slightly better. The new model achieved an accuracy of 76.32% (compared to 76.03% from the proposed model). The loss and accuracy of training and testing can be seen in Figure 8.

TABLE II: Fine Tuned VGG16 CNN Model 2

| Layers | |
|---|---|
| Input (224, 224, 3) | |
| Conv2D (224, 224, 64)<br>Conv2D (224, 224, 64) | |
| Max Pooling | |
| Conv2D (112, 112, 128)<br>Conv2D (112, 112, 128) | |
| Max Pooling | |
| Conv2D (56, 56, 256)<br>Conv2D (56, 56, 256)<br>Conv2D (56, 56, 256) | |
| Max Pooling | |
| Conv2D (28, 28, 512)<br>Conv2D (28, 28, 512)<br>Conv2D (28, 28, 512) | |
| Max Pooling | The base model was used until this layer. |
| Conv2D (14, 14, 512)<br>Batch Normalization<br>Max Pooling<br>Dropout 0.5 | The following layers were added to the network |
| Flatten | |
| Fully Connected 1024<br>Batch Normalization<br>Dropout 0.5 | |
| Fully Connected 1024<br>Batch Normalization<br>Dropout 0.5 | |
| Fully Connected 1024<br>Batch Normalization<br>Dropout 0.5 | |
| Softmax 11 | |

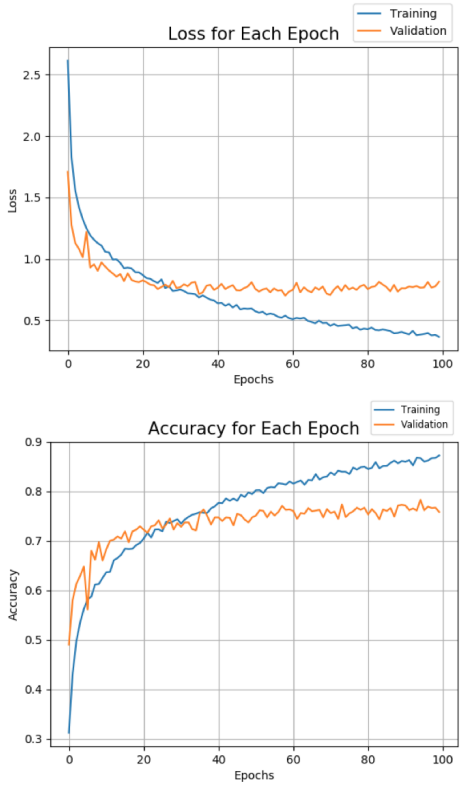


Fig. 8: Training using Model 2
Top: Training and validation loss
Bottom: Training and validation accuracy

## IV. CONCLUSION

This project presented a convolutional neural network that was trained to classify the various states of cooking ingredients in images. A network was constructed with the VGG16 network as a base model and further fine tuning was done to create the final network. The models had a tendency to overfit after the first 20 epochs so future work would be done to combat this problem. The use of other base models and more optimizers would also be explored to increase the accuracy during testing.

## REFERENCES

[1] Labelbox annotation tool. Information can be found at: https://labelbox.com.
[2] François Chollet et al. Keras. https://keras.io, 2015.
[3] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.
[4] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.
[5] Geoffrey Hinton. Neural networks for machine learning online course.
[6] Ahmad Babaeian Jelodar. Cooking state recognition challenge dataset, 2018. Dataset information can be found at: http://rpal.cse.usf.edu/datasets$_cooking_state_recognition.html$.
[7] Ahmad Babaeian Jelodar, Md Sirajus Salekin, and Yu Sun. Identifying object states in cooking-related images. *CoRR*, abs/1805.06956, 2018.
[8] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.
[9] Y. LeCun, L. Bottou, G. Orr, and K. Muller. Efficient backprop. In G. Orr and Muller K., editors, *Neural Networks: Tricks of the trade*. Springer, 1998.
[10] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.
[11] B. Zhou, A. Khosla, A. Lapedriza, A. Oliva, and A. Torralba. Learning deep features for discriminative localization. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2921–2929, June 2016.