State Classification of Cooking Images Using VGG19 Network

Rupal Agarwal
University of South Florida

***Abstract*—For the past many years, a lot of research has been going on in the field of object recognition and while we have made commendable progress in this direction, the field of object state recognition has not been explored much. This project aims to classify an image of a cooking object into one of its states using deep convolutional neural network. To train the model, transfer learning approach was used with VGG19 used as a pre-trained model. The architecture of VGG19 was slightly modified to examine the effect of different optimizers, regularizers and layers. The dataset consisted of 17 cooking objects with 11 different states. There were a total of 9309 images which were randomly divided into three sub datasets-training, validation and testing. The model was able to correctly classify the images with 70.58% accuracy on test dataset.**

***Index Terms*—object state recognition, convolutional neural network, VGG19**

I. Introduction

Nowadays, many domains like speech and audio processing, natural language processing, video games, finance, image recognition etc are using deep learning techniques to solve complex real world problems. It is because deep neural models are extremely powerful and expressive and can learn complex relationships between their inputs and output even with small datsets.

One sub domain of image recognition which is gaining popularity is food image recognition. With the help of food image recognition, robots can be trained to work in kitchens and handle food items. In order to cook, the robot should not only understand and recognize the objects such as tomato, onions but it should also be able to recognize the state of an object like chopped, whole, paste etc. [1]. Recognizing state is important as it will help the robot to understand the flow of work, guide it to grasp or handle the object in correct manner and also perform manipulation tasks. One way in which robots can learn these skills is with help of functional object-oriented network (FOON) [2]. FOON is a graphical model that represents manipulation knowledge. It defines relationship between objects and their motion so that robots can use correct object at correct time and also perform correct task on it with desired motion. It basically helps the robots to learn about object state effectively.

In order to handle a particular object, robot needs to learn about the different grasping strategies. [3] describes that robots can learn about the type of grasp and position of thumb from grasping strategies of humans. It is also important to learn about the force on fingertips while grasping an object. [4] describes a Gaussian Mixture Model (GMM) which helps to develop a force and motion model from finger tip force and position.

In the past, deep convolutional networks have been used for classification of food images. Yanai et al. [5] used deep convolutional neural networks for food image recognition. They fine tuned the model using ImageNet data to achieve classification accuracy of 78.7%. Another research was done by Hnoohom et al. in [6] to classify Thai fast food images. The convolutional deep neural model used was GoogleNet [7]. The classification accuracy on test images was about 88.3%.

This project deals with accurately recognizing the state of a cooking object like diced, cooked, paste etc. After state recognition, cooking robots will be able to decide which task to perform next with that object, identify the correct grasping strategy for the object, decide which tools to use on that object (for example- if tomato is in whole state then perhaps a knife is needed to cut it. On the other hand, if tomato is in paste state then a spoon is needed to stir it in a pan) and also how to use them (for example- if potato is in whole state then while cutting it into small pieces, knife motion will be different but while peeling them, knife motion will change).

This project exploits the VGG19 convolutional pre-trained model for food state classification task. This model was fine-tuned and trained in two steps. Different parameters such as optimizers, numbers of layers and batch size were changed and their effect on the model was analyzed. To reduce over-fitting, various regularization techniques were applied to the model like data augmentation, dropout, batch normalization, l2 regularization and early stop. The best model achieved a classification accuracy of 72.9% on validation dataset and 70.58% on test dataset.

II. Dataset

The dataset used for this project consisted of 9309 images. It was obtained from the Cooking State Recognition Challenge version 1.2 [8]. It included images of 17 cooking objects (chicken, beef, onion, tomato, bread etc.) with 11 different states (whole, sliced, chopped, peeled, julienne, grated, juice, mixed etc).

A. Data Annotation

For data preparation, data annotation was done. Every student was assigned an object (for example-tomato) and given one batch of videos of approximately 5 to 20 minutes and 0.5 to 1 frames per second. In each video, we had to identify the object and make a bounding box around it. The bounding box

Fig. 1: Food objects with different states - whole, julienne and juiced

was then labelled with a state like whole, sliced, diced etc. This process was then repeated for other frames as well. The tool used for data annotation was VATIC.

B. Data Partitioning

The whole dataset had been randomly divided into three sub datasets-training, validation and testing. Training dataset contained 6348 images and validation dataset contained 1377 images. These sub datasets were completely disjoint having no images in common. Figure 1 shows food items with different states (from left to right).

III. Model Architecture and Implementation

In order to classify states of cooking objects we used the transfer learning approach. Mostly, real world applications in deep learning do not have sufficient training data. Therefore, to overcome this problem and get good results, transfer learning [9] is used. In this approach, we use a pre-trained model along with its weights to solve our problem. For this project VGG19 neural model was used and then it was slightly modified or fine tuned with different parameters to classify the states effectively. In this section, the architecture of proposed model is discussed.

A. Base Model

The pre-trained model used for this project was VGG19 neural network.The VGG architecture was introduced by Visual Geometry Group in University of Oxford [10]. It has 19 weight layers which have been pre-trained on ImageNet.It has sixteen 3x3 convolutional layers which are placed on top of each other with three fully connected layers. It produces good accuracy in image classification.

B. Proposed Model

In the proposed model, the top three dense (fully connected) layers of the original VGG19 had been removed and instead, four new dense layers had been added. These dense layers had size of 512, 512, 256 and 11 respectively. Figure 2 shows the architecture of proposed model.

After removing top three dense layers from original VGG network, a batch normalization layer was added after the Max Pooling2D layer of block 5. Batch Normalization helps to normalize the activations of each layer by making the inputs and mean move closer to zero. In this way, the covariance shift is reduced. It has other advantages as well such as-reducing the number of epochs required, speeding up the learning process by making the model converge faster and giving regularization effect to the model by reducing overfitting.

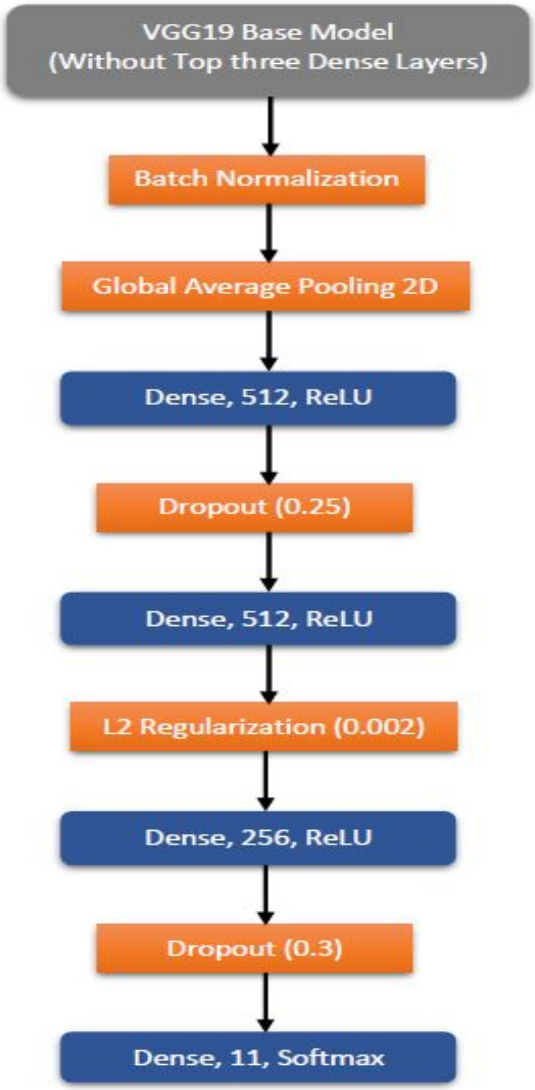

Fig. 2: Architecture of Proposed Model

The dense layers of model require a linear feature vector whereas the output of last convolutional layer is not one dimensional. Therefore, a Global Average Pooling 2D layer was added after the Batch Normalization layer to reduce the dimensionality of three dimensional tensor. It converts input tensor of shape hxwxd to 1x1xd. In addition to reducing dimensionality, this layer also helps in reducing overfitting in model as the number of parameters are greatly reduced.

While the convolutional layers help in feature extraction (features like edges, blobs, shapes etc), the classification of data into various classes is done with the help of fully connected layers. For this purpose, after the convolutional layers four fully connected layer have been added to the network. The first three layers are of size 512, 512 and 256 respectively, whereas the last layer is of size 11 which represents the total number of output (target) classes. In a fully connected layer, each input neuron is connected to each and every output

neuron.

Activation Function 'Rectified Linear Unit (ReLU)' is used for all the convolutional layers and the dense layers (except the last dense layer with size 11). If input is negative then it returns 0, whereas for positive inputs it returns that positive value. It is represented by the following formula-

$$f(x) = max(0, x)$$

In comparison to other activation functions like tanh and sigmoid, ReLU performed better for this model. It helped the network to converge faster by speeding up the computations and also removed vanishing gradient problem.

To overcome the problem of overfitting, two dropout layers were added after the first dense layer of size 512 and after dense layer of size 256. Dropout helps to randomly drop some neurons along with their connections from the neural model while training [11]. It makes the model more robust because some neurons are randomly removed and this in turn reduces the dependency among neurons.

In the second dense layer of size 512, another regularization technique called L2 regularization was added to reduce overfitting. It helps to generalize the model by reducing the value of weights.

In the last dense layer, Softmax is used as activation function. Softmax function is usually used in the last layer of neural model as it gives the probability of each target class. Out of all probabilities, the predicted class has highest one. Its output is basically a range of values between 0 and 1 [12]. Figure 3 shows the detailed layer by layer architecture of the proposed model along with output shape and number of parameters.

C. Implementation

The proposed model was implemented using Python programming language with Keras and Tensorflow.

IV. Methodology

This section describes the preprocessing operations applied on the dataset, the training process and also the input used and output produced.

A. Data Preprocessing

Before the training process, input data was processed a bit so that model achieves good classification result.

1) Data Augmentation: The original VGG19 network [10] had been trained on millions of images. But in this project there were smaller number of training images. Therefore, to exploit the full power of VGG19, number of training samples were increased. This was achieved with help of data augmentation. A number of different transformations such as rotation (15 degree), re-scaling (1/255), zooming (20%), horizontal flip (True), Vertical Flip (True), horizontal and vertical shift (20%) were applied to the input training dataset. It helps to make the model more robust and also reduces the problem of overfitting. Figure 4 shows an example of data augmentation.

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	(None, 150, 150, 3)	0
block1_conv1 (Conv2D)	(None, 150, 150, 64)	1792
block1_conv2 (Conv2D)	(None, 150, 150, 64)	36928
block1_pool (MaxPooling2D)	(None, 75, 75, 64)	0
block2_conv1 (Conv2D)	(None, 75, 75, 128)	73856
block2_conv2 (Conv2D)	(None, 75, 75, 128)	147584
block2_pool (MaxPooling2D)	(None, 37, 37, 128)	0
block3_conv1 (Conv2D)	(None, 37, 37, 256)	295168
block3_conv2 (Conv2D)	(None, 37, 37, 256)	590080
block3_conv3 (Conv2D)	(None, 37, 37, 256)	590080
block3_conv4 (Conv2D)	(None, 37, 37, 256)	590080
block3_pool (MaxPooling2D)	(None, 18, 18, 256)	0
block4_conv1 (Conv2D)	(None, 18, 18, 512)	1180160
block4_conv2 (Conv2D)	(None, 18, 18, 512)	2359808
block4_conv3 (Conv2D)	(None, 18, 18, 512)	2359808
block4_conv4 (Conv2D)	(None, 18, 18, 512)	2359808
block4_pool (MaxPooling2D)	(None, 9, 9, 512)	0
block5_conv1 (Conv2D)	(None, 9, 9, 512)	2359808
block5_conv2 (Conv2D)	(None, 9, 9, 512)	2359808
block5_conv3 (Conv2D)	(None, 9, 9, 512)	2359808
block5_conv4 (Conv2D)	(None, 9, 9, 512)	2359808
block5_pool (MaxPooling2D)	(None, 4, 4, 512)	0
batch_normalization_1 (Batch	(None, 4, 4, 512)	2048
global_average_pooling2d_1 (	(None, 512)	0
dense_1 (Dense)	(None, 512)	262656
dropout_1 (Dropout)	(None, 512)	0
dense_2 (Dense)	(None, 512)	262656
dense_3 (Dense)	(None, 256)	131328
dropout_2 (Dropout)	(None, 256)	0
dense_4 (Dense)	(None, 11)	2827

Total params: 20,685,899
Trainable params: 660,491
Non-trainable params: 20,025,408

Fig. 3: Detailed Architecture of the Proposed Model

2) Resizing Input Image: Original VGG19 model takes input image size of 224x224x3. But due to space constraint and complex model architecture, the size of image for proposed model was reduced to 150x150x3. Even though the size of input image was changed, the model still performed very well and gave quite good classification results. An even smaller

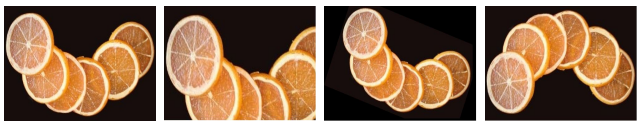

Fig. 4: Data Augmentation in images (from left to right)- original, zoom, rotation, vertical flip

image size was also tried but then the training accuracy was low. It is because if image size is very low then all the features of image are not properly extracted.

B. Input and Output

Images of size 150x150x3 were fed as input to the model and the output was a linear vector representing the probabilities of all the target classes. The index of maximum probability gave the predicted class.

C. Model Training

The training of model was done in two parts. First, all the layers below the dense layer of the proposed model were freezed and only the dense layers were trained. The learning rate for this step was kept at 0.001 with 50 epochs. In the second step of the training process, the last two blocks of VGG19 (block 4 and block 5) were unfreezed. Along with dense layers, these two blocks were also trained using the model of first part. In this step, the learning rate was reduced to 0.0001 (10% of previous learning rate). The number of epochs were again 50 and the number of trainable parameters were 660,491.

To deal with overfitting, callback (early stop and model checkpoint) mechanism was also used with patience of 5 epochs. The parameter which was monitored for early stopping was validation accuracy. The best model was saved based on the validation accuracy. This approach helps in saving the most recent best model because in a particular epoch if the validation accuracy does not improve then the previous best model is not overwritten.

For both the training parts, Adam was used as the optimizer with a decay value of 0.001. If the validation loss increased during the training process, then decay parameter would reduce the learning rate. The batch size for training samples was 100 (with 60 batches in an epoch) and 60 for validation samples (with 20 batches in an epoch). The type of loss used was 'categorical_crossentropy'. This type of loss is used when the number of output classes are more than two.

Different parameters were altered such as type of optimizers, different batch sizes, number of layers trained etc. The effect of each of these alterations have been detailed in the next section.

V. Model Evaluation and Results

In an effort to achieve high classification accuracy, different training parameters were altered and their overall effect was analyzed. Various types of modifications that were made were- trying different optimizers (Adam, SGD and RMSprop), trying different batch sizes (128, 100 and 64), and also varying number of trainable layers. In all these experiments, number of epochs were 50 with patience of 5 epochs.

TABLE I: Comparison between Training and Validation accuracy for different optimizers

Optimizer	Training Accuracy	Validation Accuracy
Adam	98.9%	72.9%
SGD	23%	29%
RMSprop	77.7%	68.2%

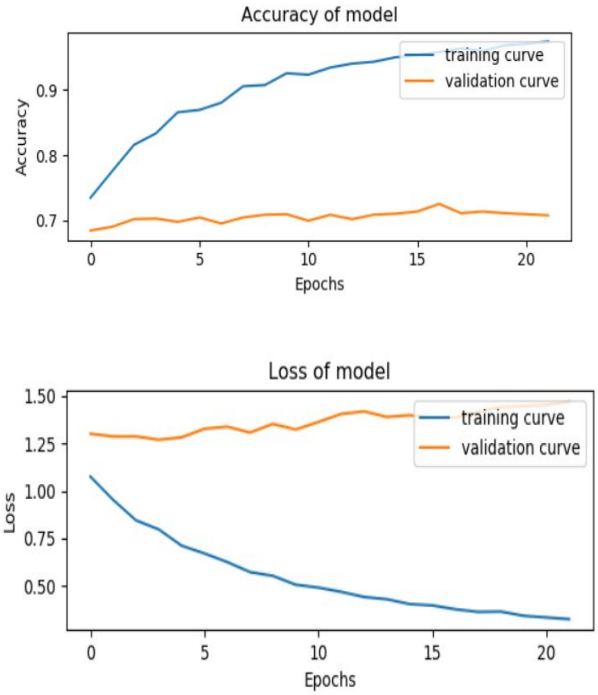

Fig. 5: Accuracy (top) and Loss (bottom) with Adam optimizer

A. Effect of different Optimizers

The proposed model was trained with three different optimizers- Adam, SGD and RMSprop. The model gave best accuracy with Adam and least accuracy with SGD as shown in Table I. In the case of SGD optimizer, network progressed very slow. Even after 50 epochs, training and validation accuracy could not increase that much. In all the three cases, the learning was 0.001 for first part of training and 0.0001 for second part of training. Figure 5, Figure 6 and Figure 7 give graphical representation of validation and training accuracy and loss for different optimizers Adam, SGD and RMSprop respectively.

B. Effect of different batch sizes

Different batch sizes were tried and tested like-128, 100 and 64 out of which batch size 100 gave the best classification accuracy for our model. Table II shows the accuracy and loss for different batch sizes. Again for all the three experiments, number of epochs were 50, learning rate for first part of training was 0.001 and for second part of training was 0.0001.

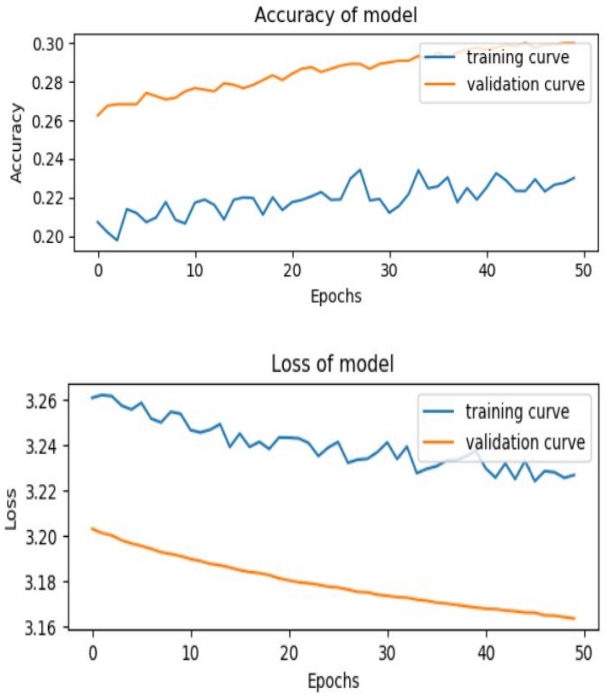

Fig. 6: Accuracy (top) and Loss (bottom) with SGD optimizer

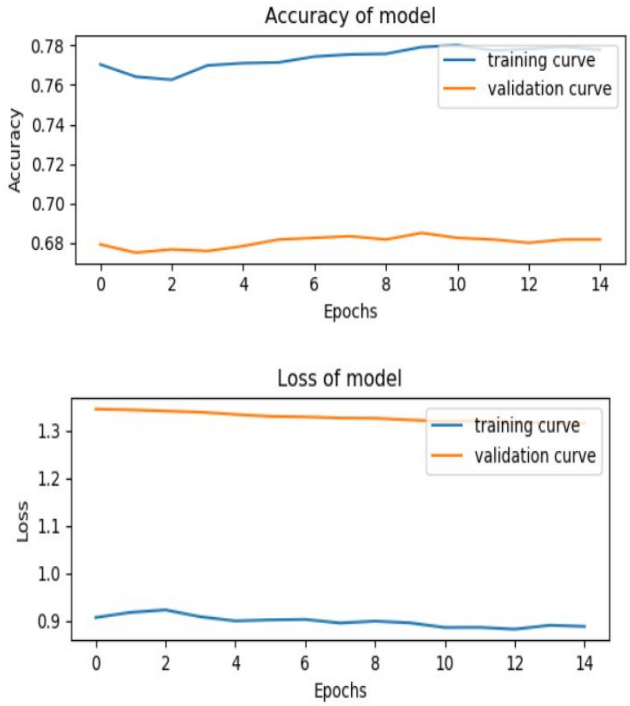

Fig. 7: Accuracy (top) and Loss (bottom) with RMSprop optimizer

TABLE II: Comparison between Training and Validation accuracy for different batch sizes

Batch Size	Training Accuracy	Validation Accuracy
128	80.7%	67.2%
100	98.9%	72.9%
64	84.3%	68.1%

C. *Effect of different trainable layers*

This experiment was done in two steps. First, all the convolutional layers of the model (block 1 to block 5) were freezed. It means that their weights were not allowed to change and instead the weights of pre-trained VGG model were used. Only the newly added last four dense layers were allowed to be trained. In second step, block 4 and block 5 of base model (model used in first step) were unfreezed and they were also trained along with top dense layers. By doing this, a higher classification result was achieved. It is because in the first step, the number of trainable parameters were very less. Also, originally, VGG19 network was trained on Image Net dataset and its weights were set according to that dataset. Every dataset has its own unique properties, therefore it is always better to fine tune a pre-trained model with the dataset that we are going to use. Table III shows the comparison between training and validation accuracy when different number of layers were freezed and Figure 8 shows the graphical view of accuracy and loss when all the convolutional layers of model were freezed.

TABLE III: Comparison between Training and Validation accuracy for different layers

Layer	Training Accuracy	Validation Accuracy
All freezed (except dense layers)	93.1%	66.8%
Block 4 & 5 unfreezed along with dense layers	98.9%	72.9%

VI. Conclusion and Future Work

In this project, a pre-trained convolutional model- VGG19 was fine tuned to classify states of food images into diced, sliced, whole etc. Various parameters such as different optimizers, different batch sizes, and different layers were modified and their effect on accuracy was observed. The best model was able to achieve a classification accuracy of 72.9% on the validation dataset and about 70.58% accuracy on test dataset. While fine-tuning the weights of model, different regularization techniques were applied to reduce overfitting. Techniques like data augmentation, l2 regularization, batch normalization, dropout etc. were used, but still the model was slightly overfitted. This can be because of several reasons. First, some images in dataset belonged to different states but they appreared quite similar. Figure 9 shows one such example. Second, instead of validation accuracy, validation loss should have been monitored for saving the best model. It is because while training the model, validation accuracy dropped but the validation loss increased in some epochs. Therefore, a check should have been placed on the validation loss. Third, it is

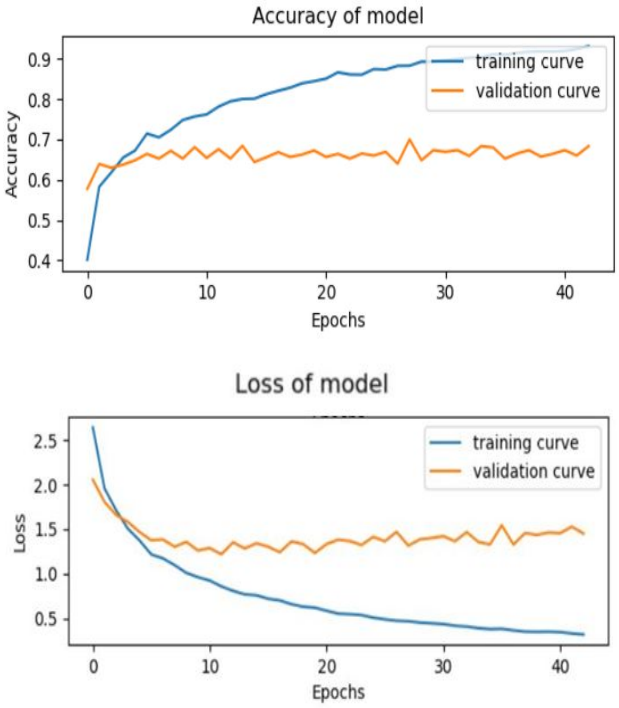

Fig. 8: Accuracy and Loss after freezing all convolutional layers

Fig. 9: Example of an image which can be classified as whole or creamy-paste

possible that some important features may have been lost while training because for this model image size was downsized to 150x150x3, but VGG19 model was originally trained on image size 224x224x3.

Hence, as a part of future work, this model can be trained with the original image size on a more powerful hardware having increased storage space. Different architectures can also be tried like Inception V3 [13] or Resnet [14]. In addition to this, more training images can also be used as it can reduce overfitting and improve classification accuracy.

REFERENCES

[1] A. B. J. Md Sirajus Salekin and R. Kushol, "Cooking state recognition from images using inception architecture," in *2019 International Conference on Robotics,Electrical and Signal Processing Techniques (ICREST)*, 2019, pp. 163–168.

[2] Y. Sun, "Ai meets physical world – exploring robot cooking," in *arXiv:1804.07974v1 [cs.RO]*, 21 Apr 2018.

[3] Y. Lin and Y. Sun, "Grasp planning based on strategy extracted from demonstration," in *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2014, pp. 4458–4463.

[4] M. C. Yun Lin, Shaogang Ren and Y. Sun, "Learning grasping force from demonstration," in *2012 IEEE International Conference on Robotics and Automation*, 2012, pp. 1526–1531.

[5] K. Yanai and Y. Kawano, "Food image recognition using deep convolutional network with pre-training and fine-tuning," in *2015 IEEE International Conference on Multimedia & Expo Workshops (ICMEW)*, 2015, pp. 1–6.

[6] N. Hnoohom and S. Yuenyong, "Thai fast food image classification using deep learning," in *2018 International ECTI Northern Section Conference on Electrical, Electronics, Computer and Telecommunications Engineering (ECTI-NCON)*, 2018, pp. 116–119.

[7] Y. J. P. S. S. R. D. A. D. E. V. V. Christian Szegedy, Wei Liu and A. Rabinovich, "Going deeper with convolutions," in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015, pp. 1–9.

[8] M. S. S. Ahmad Babaeian Jelodar and Y. Sun, "Identifying object states in cooking-related images," in *arXiv:1805.06956v3 [cs.CV]*, 30 Oct 2018.

[9] T. K. W. Z. C. Y. Chuanqi Tan, Fuchun Sun and C. Liu, "A survey on deep transfer learning," in *arXiv:1808.01974v1 [cs.LG]*, 6 Aug 2018.

[10] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *arXiv:1409.1556v6 [cs.CV]*, 10 Apr 2015.

[11] A. K. I. S. Nitish Srivastava, Geoffrey Hinton and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," in *The Journal of Machine Learning Research, vol. 15, no. 1*, 2014, pp. 1929–1958.

[12] A. G. Chigozie Enyinna Nwankpa, Winifred Ijomah and S. Marshall, "Activation functions: Comparison of trends in practice and research for deep learning," in *arXiv:1811.03378v1 [cs.LG]*, 8 Nov 2018.

[13] S. I. J. S. Christian Szegedy, Vincent Vanhoucke and Z. Wojna, "Rethinking the inception architecture for computer vision," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 2818–2826.

[14] S. R. Kaiming He, Xiangyu Zhang and J. Sun, "Deep residual learning for image recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778.