Classifying Object States in Cooking-Related Images Using Fine-tuned VGG19 Networks

Zaid Marji

***Abstract*—This paper summarizes the task of object state classification for cooking-related images, and introduces the reader to the architecture used in VGG19 convolutional neural network. Our work builds upon the VGG19 network by using its architecture as a starting point, and we use the ImageNet weights to initialize the neural network's weights for the applicable layers. And finally, we will discuss how the VGG19 was modified and trained to obtain our results.**

***Index Terms*—Object State Classification, Transfer Learning, Fine-tuning Neural Networks, VGG, Convolutional Neural Networks**

I. Introduction

A. Motivation

In the fields of artificial intelligence, and its applications in robotics in particular, understanding images and videos is a critical endeavor on the path towards creating more useful and human-like robots. In this paper, we discuss our work towards the task of classifying object states in cooking-related images. The main long-term goal of this task is to create robotic machinery that is capable of manipulating its environment [6]. In this paper, we address one specific challenge that requires learning how to manipulate the environment, namely, cooking a simple meal. While this goal is trivial for a human to perform, the state of the art technology and the related theoretical underpinnings are currently lacking for the goal of robotic machinery to be able to perform this simple task. One of the challenges that needs to be addressed to reach our eventual goal is to enable the robotic machinery to understand its environment. In particular, for our task we need the robot to recognize everyday objects (eg. knife, spoon, egg, tomato, cabinet, refrigerator, ...etc) as well as the states of those objects (eg. whole tomato, sliced egg, ...etc). The task of recognizing objects has been researched extensively [6], [15], [16], [4]. On the other hand, the task of recognizing the state of such objects is not as extensively researched [6]. In this paper, we discuss our work on the latter task of identifying the state of objects in the environment, especially those states that are relevant to the task of cooking a simple meal.

B. Background on the object states classification task

In order to enable a robot to cook a simple meal, we need to equip the robot with an understanding of its environment. This understanding must include recognizing which objects are present in their environment, as well as, the state of those objects. A state can be defined as an observable characteristic of objects which can be described in terms of form, texture, or color [6]. A typical example would be observing the state of a tomoato. A tomato could be in any of many states, including: sliced, diced, or whole. Consider, for example, a scenario where a robot wants to prepare a salad. In this scenario, the robot will need to obtain a whole tomato, wash it, slice it, and finally dice it. In order to perform those tasks, the robot will need to first recognize that a tomoato is present in its environment, and that the tomato is in a whole state. Alternatively, it is possible that diced tomato is present in the kitchen, and the robot will only need to use the pre-prepared tomatoes. Recognizing which objects are present, and their current state is vital to this task. [6]

One reason why state recognition is necessary is that the robot might need to handle the same object in different states differently. The way a robot needs to pick up a whole tomato is going to be different from the way the robot needs to pick up a sliced tomoato, and yet again different from the way the robot needs to pick up a diced tomoato. These differences necessitate the use of state recognition while performing the task of cooking a simple meal [6], [7], [10], [8], [9].

Another issue that is relevant for the task of preparing a simple meal is how actions that the robot can perform transform the state of an object. Consider, for example, using a knife to slice a whole tomato. This action transforms the tomato object from the whole state to the sliced state. In order to cook a simple meal, cooking instructions usually require items to be in certain states. For example, in order to prepare a salad, we need to use diced tomatos. The robot will need to figure out the current state of the tomato, and its target state, and plan its actions accordingly [6].

Figure 1 shows the range of states that are typically encountered in the challenge of cooking a meal. The gray states have been selected as a representative sample of the states that will be explored in this task [6].

C. Background on transfer learning and fine-tuning artificial neural networks

Transfer learning is an action where we use learning models from one domain in another domain. Transfer learning is also sometimes called domain adaptation. The idea here is that we exploit what a model has learned in one domain to improve the generalization and accelerate the learning in another domain. [3]

In artificial neural networks (ANNs), transfer learning is done by using the learned weights in one ANN to initialize the weights in another ANN. This requires that the two models share at least some parts of their architecture, usually, the earlier layers. When training the ANN you have three possible

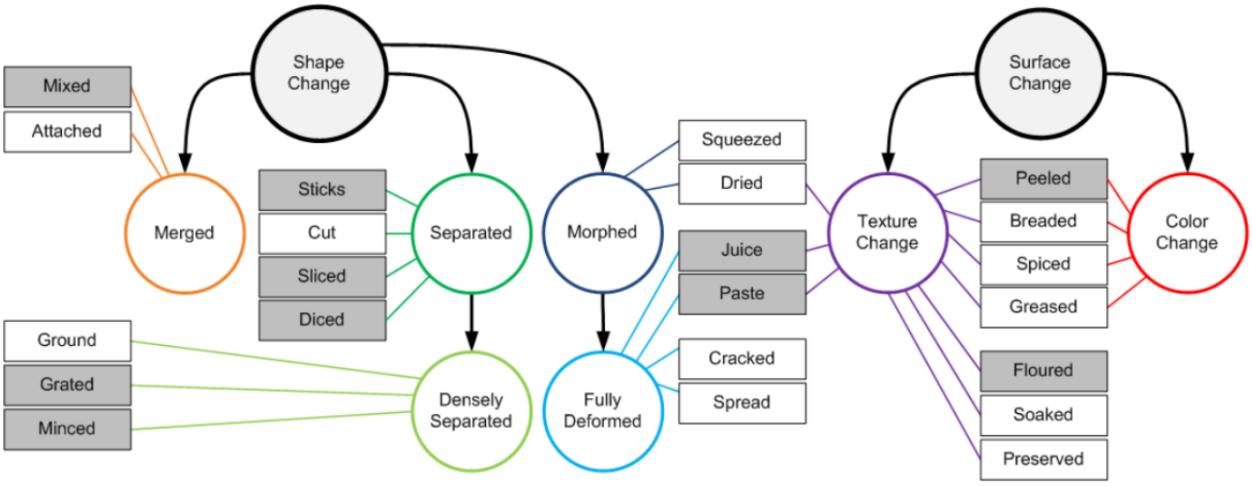

Figure 1. State identification problem definition [6]

alternatives: 1- Training only the randomly initialized layers, leaving the transfered weights unmodified. 2- Training all the layers and all the weights in the network. 3- Training part of the tranfered weights, while leaving others unmodified. Alternatives (2) and (3) that involve training some or all of the transfered weights in the network is usually called FINE-TUNING the network.

A common practice when fine-tuning an ANN is to do it in two stages. In the first stage, we only train the randomly initialized weights. In the second stage, we include the additional layers that we want to train. This is done in order to allow the network to find good weights for the randomly initialized weights before modifying any of the transfered weights. Fine-tuning can improve the performance of an ANN by allowing the transfered weights to be modified to better suit the the domain that the ANN is learning.

D. Background on the VGG19 architecture

Since 2010, the annual ImageNet Large Scale Visual Recognition Challenge (ILSVCR) [2], commonly called the ImageNet challenge, is a competition where research teams submit programs that classify and detect objects and scenes. Since then, numerous approaches and architectures have been proposed and attempted to compete in the ImageNet challenge. [13]

VGG Net [12] is a convolutional neural network architecture proposed by the Visual Geometry Group (VGG) from University of Oxford for the ImageNet challenge of 2013. VGG Net didn't win the ImageNet 2013 challenge but it is still used by many people because it was a simple architecture [13]. This architecture is characterized by using deep CNNs of depths typically in the range of 16-19, and it uses very small (3x3) convolution filters [12].

Table I shows several of the VGG Net architectures that were proposed. Configuration D is commonly referred to as VGG16, while configuration E is commonly referred to as VGG19. In all of the proposed architectures, a stack of convolutional layers is followed by three fully-connected layers. The first two contain 4096 neurons per layer, while the last one contains 1000 neurons matching the number of classes that the ImageNet uses. The final layer is the soft-max layer. All of the hidden layers use a ReLU non-linear rectifier. [12]

Table I
VGG NET CONFIGURATIONS (SHOWN IN COLUMNS). [12]

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

The depth of the configurations increases from the left (A) to the right (E), as more layers are added (the added layers are shown in bold). The convolutional layer parameters are denoted as "conv<receptive field size>-<number of channels>". The ReLU activation function is not shown for brevity.

The VGGNet creators provided their intuition behind their design choices for their proposed architecture. Namely, that having 2 consecutive 3x3 filters gives an effective receptive field of 5x5, and 3 consecutive 3x3 filters give a receptive field

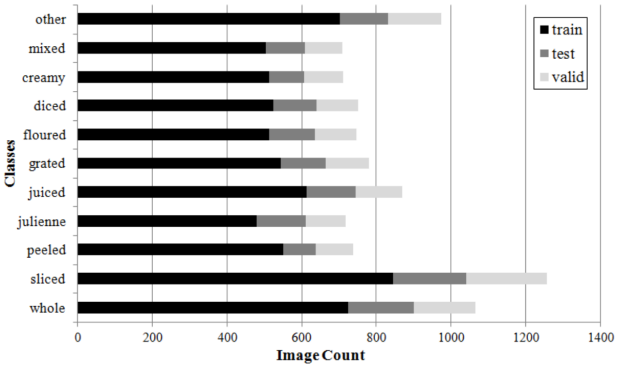

Figure 2. The dataset statistics [6]

of 7x7 filters [12]. By using this architecture, we create a CNN with much fewer parameters that need to be trained. As you may notice from the architecture the number of filters double after every max-pooling operation. As a data augmentation technique scale jittering was used. It was trained with batch gradient descent and used ReLUs. [13]

II. Dataset creation and description

A. Where to find the dataset

The dataset that is used can be found at the University of South Florida's Robot Perception and Action Lab website which can be found at http://rpal.cse.usf.edu/datasets_cooking_state_recognition.html. The dataset version 1.2 is the specific dataset that was used for training, validating, and testing the CNN model.

B. How the dataset was created

The dataset images were obtained by searching through the Google search engine using a keyword combination of each object and state (for example, "sliced tomato"). The URLs of found images were downloaded and screened. Then the images were published on a local server and dispersed for manual labeling using the Vatic annotation tool [14]. The labels were further reviewed and stored in their designated classifications within the dataset. Some of the images was labeled using the labelbox tool [1]. [6]

The resulting dataset consisted of 17 cooking objects, and 11 classes of states (whole, peeled, floured, sliced, diced, grated, julienne, juice, creamy, mixed, other). The dataset contains 9309 images. Figure 2 shows the statistics of each class in the dataset. Figure 3 shows examples of images from each of the 11 classes in the dataset. [6]

C. The object state classifications

The object states are classified into 11 classes. Those classes are as follows: [6]

1) The WHOLE state: It contains objects in their original format and shape (see Figure 3, column 1).
2) The PEELED state: It contains objects that are peeled but not cut, sliced, or morphed (see Figure 3, column 2).
3) The FLOURED state: It contains objects that are floured (see Figure 3, column 3).
4) The GRATED state: It contains objects that are densely separated, such as bread crumbs, minced garlic, or grated egg (see Figure 3, column 4).
5) The JULIENNE state: It contains objects such as carrot sticks, French fries, julienne pepper, or shredded meat (see Figure 3, column 5).
6) The DICED state: It contains diced or chopped objects (see Figure 3, column 6).
7) The SLICED state: It contains objects that are thinly-sliced. Objects that are cut in other ways (such as cut in half or diced) are not considered as sliced. (See Figure 3, column 7.)
8) The JUICED state: It contains objects such as milk, melted butter, and tomato juice (see Figure 3, column 8).
9) The CREAMY state: It contains objects that are creamy, such as cream, creamy butter or cheese, garlic or tomato paste, and mashed potato (see Figure 3, column 9).
10) The MIXED state: It contains a scramble of multiple objects such as salads (see Figure 3, column 10).
11) The OTHER state: It contains any state that is not listed in any of the previous states (see Figure 3, column 11).

Note that not all states are applicable to all objects. Objects have a specific set of allowed classifications [6].

III. Methodology

A. The base model

The VGG19 architecture is used as the base model. This model was chosen for its simplicity, as well as, its efficacy as a base model.

B. The proposed model architecture

The model architecture that is used uses all the layers in the VGG19 model including first the two fully-connected layers of size 4096, except the last layer that is the softmax layer which is removed. On top of the last fully-connected layer a dropout layer with dropout factor of 0.5 was added, followed by a fully-connected dense layer composed of 512 neurons, followed by another dropout layer with dropout factor of 0.5, and finally an output softmax layer composed of 11 neurons.

Table II shows the a detailed view of the proposed model. Note that the transfered column indicates whether or not a given layer was transfered from the VGG19 base model, and its weights initialized using the ImageNet weights.

Table II shows all the layers in the proposed model, and which layers were transfered from the base model without training, which layers were transfered from the base model and trained, and which layers are unique to our proposed model.

C. Data augmentation

Since the sample data is not large enough for the size of the convolutional neural network that we are training, we opted to augment the data. There are several data augmentation techniques, including: Rescaling, rotation, width shifting, height

Figure 3. Examples of images in the dataset in each of the 11 classes [6]

Table II
OVERVIEW OF THE PROPOSED MODEL

Layer (type)	Parameters	Transferred	Trained
block1_conv1 (Conv2D)	1792	Yes	No
block1_conv2 (Conv2D)	36928	Yes	No
block1_pool (MaxPooling2D)	0	Yes	N/A
block2_conv1 (Conv2D)	73856	Yes	No
block2_conv2 (Conv2D)	147584	Yes	No
block2_pool (MaxPooling2D)	0	Yes	N/A
block3_conv1 (Conv2D)	295168	Yes	No
block3_conv2 (Conv2D)	590080	Yes	No
block3_conv3 (Conv2D)	590080	Yes	No
block3_conv4 (Conv2D)	590080	Yes	No
block3_pool (MaxPooling2D)	0	Yes	N/A
block4_conv1 (Conv2D)	1180160	Yes	No
block4_conv2 (Conv2D)	2359808	Yes	No
block4_conv3 (Conv2D)	2359808	Yes	No
block4_conv4 (Conv2D)	2359808	Yes	No
block4_pool (MaxPooling2D)	0	Yes	N/A
block5_conv1 (Conv2D)	2359808	Yes	No
block5_conv2 (Conv2D)	2359808	Yes	No
block5_conv3 (Conv2D)	2359808	Yes	Yes
block5_conv4 (Conv2D)	2359808	Yes	Yes
block5_pool (MaxPooling2D)	0	Yes	N/A
flatten (Flatten)	0	Yes	N/A
fc1 (Dense)	102764544	Yes	Yes
fc2 (Dense)	16781312	Yes	Yes
dropout_1 (Dropout)	0	No	N/A
dense_1 (Dense)	2097664	No	Yes
dropout_2 (Dropout)	0	No	N/A
predictions (Dense)	5643	No	Yes

Table III
DATA AUGMENTATION TECHNIQUES AND THEIR PARAMETERS

Name	Parameter
Rescale	Not used
Rotation	Not used
Width shifting	Not used
Height shifting	Not used
Shear	0.2
Zoom	0.2
Horizontal flipping	True
Vertical flipping	Not used

shifting, shear, zoom, horizontal flipping, and vertical flipping. We opted to use only a subset of those data augmentation techniques. Table III shows the various data augmentation techniques and their parameters.

D. Data preprocessing

DATA PREPROCESSING is a technique that involves transforming raw data into a manageable and consistent format. In the domain of image classification, the raw image data needs to be preprocessed before it is used as an input to the CNN [11], [5]. The only data preprocessing that is done in the VGG19 architecture is subtracting the mean RGB value, computed on the training set, from each pixel [12]. Since we are using the weights from the VGG19 ImageNet pre-trained model, we use the same preprocessing operation as provided by the VGG19 pre-trained model.

E. Training

1) Transfer learning: As shown in Table II, the model's weights were initialized using the VGG19 ImageNet pre-trained model.

2) Fine-tuning: As shown in Table II, the fully-connected layers, as well as, the last two convolutional layers of the VGG19 ImageNet pre-trained model were fine-tuned.

3) Early stopping: A unique early stopping algorithm has been used while training the model. The advantages or disadvantages of this early stopping algorithm have not been

thoroughly investigated, however, this algorithm was designed to be robust to fluctuations in the validation accuracy. Algorithm 1 shows the logic that was used to terminate a training session. The algorithm will run the training operation for 10 epochs at a time, recording the current validation accuracy, the best validation accuracy, and the previous validation accuracy. Every time the current validation accuracy is less than the previous validation accuracy, a counter is incremented. The counter is reset every time the best validation accuracy is updated. Finally, a training session ends when the counter reaches three.

Algorithm 1 Early stopping algorithm

```
N = 3                          ▷ This is a modifiable parameter
epochs = 10                    ▷ This is a modifiable parameter
BestAccuracy = ValidationAccuracy(model)
LastAccuracy = ValidationAccuracy(model)
DegradationCount = 0
while DegradationCount < N do
    TrainModel(model, epochs)
    CurrentAccuracy = ValidationAccuracy(model)
    if CurrentAccuracy >= BestAccuracy then
        BestAccuracy = CurrentAccuracy
        DegradationCount = 0
    else if LastAccuracy >= CurrentAccuracy then
        DegradationCount += 1
    end if
    LastAccuracy = CurrentAccuracy
end while
```

4) Training schedule: The training has been done in 9 stages as follows:

1) Using ADAM optimizer with learning rate of 1e-3 and training only the randomly initialized layers.
2) Using ADAM optimizer with learning rate of 1e-4 and training only the randomly initialized layers.
3) Using ADAM optimizer with learning rate of 1e-5 and training only the randomly initialized layers.
4) Using ADAM optimizer with learning rate of 1e-5 and training all the randomly initialized layers and the first fully-connected layer below.
5) Using ADAM optimizer with learning rate of 1e-5 and training all the randomly initialized layers and both of the fully-connected layers below.
6) Using ADAM optimizer with learning rate of 1e-5 and training all the randomly initialized layers, both of the fully-connected layer below, and the first convolutional layer below.
7) Using ADAM optimizer with learning rate of 1e-5 and training all the randomly initialized layers, both of the fully-connected layers below, and the first two convolutional layers below.
8) Using ADAM optimizer with learning rate of 1e-6 and training all the randomly initialized layers, both of the fully-connected layers below, and the first two convolutional layers below.
9) Using SGD optimizer with learning rate of 1e-5 and training all the randomly initialized layers, both of the fully-connected layers below, and the first two convolutional layers below.

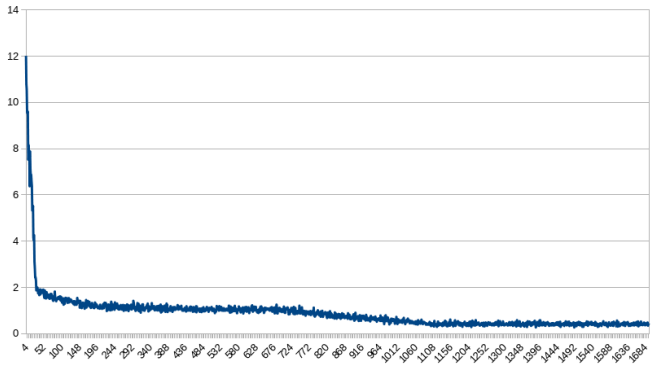

Figure 4. Training loss of the proposed model

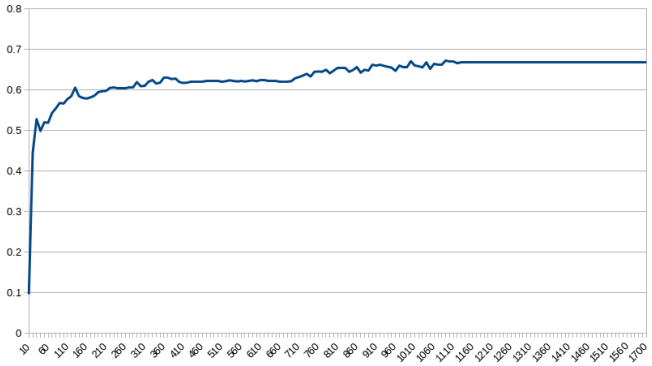

Figure 5. Validation accuracy of the proposed model

IV. Evaluation and Results

The proposed model has an accuracy of 81.1% on the validation dataset, and an accuracy of 81.9% on the testing dataset. Figure 4 shows the loss function value on the training dataset on each epoch. The X-axis represents the epoch number, while the Y-axis shows the loss function values. As it can be observed, the loss function value decreases steadily throughout the training, showing very noticeable improvements very early in the training process. Figure 5 shows the top-1 accuracy on the validation dataset for every 10 epochs. The X-axis represents the epoch number, while the Y-axis shows the top-1 accuracy. As it can be observed, the top-1 accuracy on the validation dataset behavior mirrors that of the loss function value on the testing dataset. It starts really low, and improves drastically at the begining, and continues to improve throughout the training process.

V. Discussion

A. Overview of experiments and explorations

The following design choices were attempted in different configurations and different permutations:

- Two base models were experimented with:
  - VGG-16
  - VGG-19 *

- Various architectures were experimented with. These were some the layer choices used beyond the convolutional layers:
  - FC-4096, FC-4096, Dropout, FC-512, Dropout, Softmax-11 *
  - FC-4096, FC-4096, Dropout, FC-128, Dropout, Softmax-11
  - FC-4096, Dropout, FC-512, Dropout, Softmax-11
  - FC-4096, Dropout, FC-128, Dropout, Softmax-11
  - FC-4096, FC-4096, Dropout, Softmax-11
  - FC-4096, FC-512, Dropout, Softmax-11
  - FC-4096, FC-128, Dropout, Softmax-11
  - FC-4096, Dropout, FC-512, Softmax-11
  - FC-4096, Dropout, FC-128, Softmax-11
- Various optimizers were experimented with:
  - ADAM *
  - RMSProp
  - SGD
- Various regularizers were experimented with:
  - L_1
  - L_2
  - Dropout *
  - Early stopping *
  - Batch normalization *
- Two dropout factors were experimented with:
  - 0.2
  - 0.5 *
- Various learning rates were experimented with, including: 1e-2, 1e-3, 1e-4, 1e-5, 1e-6.

B. How the current architecture was chosen

The current architecture of the CNN model and the training schedule used is a result of many trial-and-error experiments. Obviously, not all permutations of the aforementioned design choices were experimented with as that is a very time consuming endeavor that was impractical to realize. After many trials, the design choices marked with an asterisk (*) have been chosen as the basis for further experimentation based on the observation that this combination seems to perform consistently well.

The training schedule was also developed through trial and error. The training schedule was developed by adding one stage at a time, and experimenting with different optimizers and learning rates. Once a good balance of accuracy, training speed, and good generalization is achieved, a new stage was added to get better accuracy.

VI. Conclusions

The ADAM optimizer tends to provide faster convergence and improved training speed. It has been observed that it is a good practice to start out by training the randomly initialized weights before modifying any of the weights that were transferred from the pre-trained model. It has also been observed that it is a good practice to start out with a high learning rate (ie. 1e-2 for SGD or RMSProp or 1e-3 for ADAM), and gradually decrease the learning rate. It also seems to be a good practice to gradually increase the number of layers from the pre-trained model that are fine-tuned. We have also found that a good sanity check would be to run and additional stage using the SGD optimizer with a learning rate of 1e-5.

VII. Future work

While working on this task, it seems necessary to be more systematic in our exploration of different architectures, and more thorough in their evaluations. As the project progressed, it became obvious that we need to add more logs and traces of all the experiments that were done, and be able to catalog them in an accessible manner. Going through all the experiments that were done was not an easy task, as some of the logs of earlier experiments did not include all the necessary information to study the experiment that the logs recorded. These issues were fixed over time, and they still need to be further improved to help generate reliable conclusions from the experiments that were run.

References

[1] Labelbox annotation tool kernel description. https://www.labelbox.io/, 2013.

[2] Jia Deng Jia Deng, Wei Dong Wei Dong, R. Socher, Li-Jia Li Li-Jia Li, Kai Li Kai Li, and Li Fei-Fei Li Fei-Fei. ImageNet: A large-scale hierarchical image database. *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 2–9, 2009.

[3] Goodfellow Ian, Bengio Yoshua, and Courville Aaron. *Deep Learning*. MIT Press, 2016.

[4] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. 2015.

[5] J. Herrero, R. Díaz-Uriarte, and J. Dopazo. Gene expression data preprocessing. *Bioinformatics*, 19(5):655–656, 2003.

[6] Ahmad Babaeian Jelodar, Md Sirajus Salekin, and Yu Sun. Identifying Object States in Cooking-Related Images. pages 1–7, 2018.

[7] Yun Lin and Yu Sun. Grasp planning to maximize task coverage. *International Journal of Robotics Research*, 34(9):1195–1210, 2015.

[8] Yun Lin and Yu Sun. Robot grasp planning based on demonstrated grasp strategies. *International Journal of Robotics Research*, 34(1):26–42, 2015.

[9] Yun Lin and Yu Sun. Task-based grasp quality measures for grasp synthesis. *IEEE International Conference on Intelligent Robots and Systems*, 2015-Decem:485–490, 2015.

[10] Yun Lin and Yu Sun. Task-oriented grasp planning based on disturbance distribution. *Springer Tracts in Advanced Robotics*, 114:577–592, 2016.

[11] Astha Sharma. State Classification with CNN. *arXiv preprint arXiv:1806.03973*, 2018.

[12] Karen Simonyan and Andrew Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. pages 1–14, 2014.

[13] Rajat Vikram Singh. ImageNet Winning CNN Architectures - A Review. pages 3–8, 2015.

[14] Carl Vondrick, Donald Patterson, and Deva Ramanan. Efficiently scaling up crowdsourced video annotation: A set of best practices for high quality, economical video labeling. *International Journal of Computer Vision*, 101(1):184–204, 2013.

[15] Bangpeng Yao, Xiaoye Jiang, Aditya Khosla, Andy Lai Lin, Leonidas Guibas, and Li Fei-fei. Human Action Recognition by Learning Bases of Action Attributes and Parts. 2011.

[16] Ting Yao, Yingwei Pan, Yehao Li, Zhaofan Qiu, and Tao Mei. Boosting Image Captioning with Attributes. *Proceedings of the IEEE International Conference on Computer Vision*, 2017-Octob:4904–4912, 2017.