Elijah Elrod

[[1]](#footnote-1)

State Classification with Convolutional Neural Networks

*Abstract*— Currently object recognition of object type has reached acceptable levels of human-like accuracy. Therefore, the next step is the detection of an object’s state, such as if an item such as an onion, appears whole, cut, sliced, etc. To achieve classification of object state, a convolutional neural network (CNN) will be used. The CNN will utilize layers such as convolution, dropout, maxpooling, dense, and flattening to classify object state.

# INTRODUCTION

Being able to classify not only object type but also object state is extremely important as automation replaces manual tasks. The intent is to progress robotics by improving on the ability to recognize object state. Robot’s need to be able to identify how objects must be gripped since different states require different grips. This is due to shape of the object potentially changing with the state. For example, whole onions are round, but cut onions have a flat side where some material has been cut away. These require two different

The experiments have been performed on \*\*\*\*#\*\*\*\*\* images which have been expanded from the original set of 200 images. The original set underwent augmentation during preprocessing to enrich and enlarge the original dataset due to deep learning models requiring a significant number of samples to train on. A few augmentations performed include flipping in both the vertical and horizontal direction, scaling, and rotations. During preprocessing, training images were also resized to 128x128 pixels and converted to grayscale using OpenCV. The CNN was developed using Python, and TensorFlow Keras. The Sequential API was used to create a CNN combining a variety of layers such as convolution, pooling, activation, flattening, dense, and dropout to prevent overfitting. Experiments were running across 50 epochs.

# Data & Preprocessing

Deep learning models such as CNN’s require extensive amounts of data for CNN’s to learn the patterns required for classifying object state. The original set provided was 200 frames, which were hand labeled using CVAT.org, an open source annotation tool. As the original set was only 200 frames, the original number of samples came out to be around 236 due to some images containing multiple bounding boxes. This is an issue as the size of the current dataset is not optimal for training a deep learning model such as a CNN, therefore, augmentations must be performed on the dataset prior to training. These augmentations accomplish two major goals:

1. The data set becomes expanded, giving the CNN more images and therefore, more opportunities to learn feature patterns for classifying object state.
2. The data set also becomes enriched because the augmentations provide several variations for each original image in the dataset i.e., flipped horizontally and vertically, scaled, sheared, and rotated by a varying number of degrees.

Data for the model was extracted by locating bounding boxes around the objects of interest. After the interior image of the bounding box has been extracted, we must standardize the image so that it can be fed into the model. For this experiment, the images have been resized to 128x128 pixels and converted to grayscale to improve convergence rates.

# methodology

## Proposed Model

The CNN is built using a typical layer pattern of Conv2D to perform the 3x3 kernel function, Maxpool to get the feature maximums for each 2x2 patch, and Dropout to prevent overfitting. Afterwards, the results are flattened, and feed into two Dense layers of size 64, and 11. The model is shown via the Keras Summary function of the Model API. <#Insert the image here>. Now we will proceed to explain the function of each layer within the model, from input to output. Keras allows us to easily construct models using the Sequential API by using a stack mechanism. We add layers to the stack, and pop layers to remove them. This allows for faster iteration of experimenting with different layer configurations.

# Evaluation & Results

## Experiments

The model was trained with two different optimizers namely (Adam, and SGD) for measuring speed of convergence and the ability to generalize across the data. As we will see in the next section, the optimizers perform better in different circumstances. Adam will be shown to converge faster, whereas SGD will converge slower but be better at generalizing.

## Evaluation Of Results

Tensorboard will be used to visually display the epoch-loss and epoch-accuracy for the model. After running the CNN for 50 epochs, the logs for tensorboard to evaluate epoch accuracy and epoch loss are stored. Running tensorboard locally, we can visualize the model’s progress across epochs. In the graphs below, the orange lines represent training data results, while the blue lines represent the validation data results.

# Discussion

Further improvements could be made by more finely tuning hyperparameters. Different constructions by adding or removing layers could also further improve the classification results.

References

1. [↑](#footnote-ref-1)