Project Interim Report: Object Detection using the YOLO Algorithm and a CNN classifier

Min-Gyo Jeong, G#01402966
George Mason University
Fairfax, VA, USA
mjeong6@gmu.edu

John Stephen Gutam, G#01413212
George Mason University
Fairfax, VA, USA
jgutam@gmu.edu

Abstract

In this project interim report, we describe our efforts and modifications of the plan we made after the initial project proposal. As suggested in the comment, we will implement our own CNN model to classify images. We discuss our initial CNN model for the project, explain what are the issues we are currently experiencing, and present our efforts in order to overcome the difficulties we are encountering.

Keywords

Yolo, Object Detection, segmentation, CNN, Neural Network

1 Initial Convolutional Neural Network Structure

In the initial project report, we received a comment that we are required to build our own neural network and train it on our own for a particular dataset. To accept this suggestion, we worked on building our convolutional neural network (CNN) as shown in Fig. 1. First, we resize the image to 192 x 192 with the assumption that the image is in a grayscale to simplify the neural network structure and training time.

The model specifies an architecture suitable for image classification. In the convolutional base, known as `self.feature`, there are successive convolutional layers. The first and second layers have 32 filters each of size 3×3 with padding of 1, batch normalization, and ReLU activations. Post the second convolution, a max pooling with a 2×2 window reduces the spatial dimensions by half.

The process repeats with the third and fourth convolutional layers, this time with 64 filters each. Again, batch normalization and ReLU activations are used, followed by max pooling. Consequently, the spatial dimensions are quartered relative to the original image size.

The classifier commences with a dropout layer at a rate of 0.5, ostensibly to reduce overfitting. This is followed by a linear layer, reshaping the flattened feature maps into a vector feeding into a dense layer with a predefined number of neurons set by `dense_size`. Batch normalization and another dropout with ReLU activation succeed in this layer.

Another dense layer of the same size follows, including batch normalization and dropout. A softmax activation, which

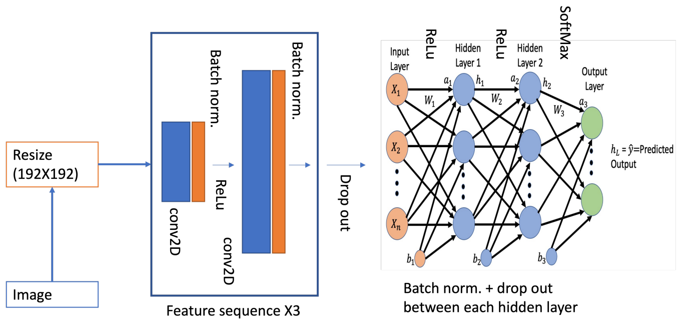

Figure 1: The Initial Convolutional Neural Network Structure

is conventionally at the network's end, is placed before the final linear layer that maps to the `num_class` classes.

2 Encountered Difficulties and Efforts to Overcome the Difficulties

While we tried to build our neural network, we encountered several difficulties. First of all, we did not have access to any CUDA GPUs, therefore, the training time took longer than expected. However, since we now have access to the GMU CPU cluster, we believe this can be mitigated in the following weeks. Second, we tried to find several datasets openly available. However, due to our lack of experience, loading datasets with different structures into the custom model was challenging and it caused a lot of errors. We have not decided which image dataset we will use in our final implementation, we are trying to find the most recent image dataset with a broader range of class labels. We believe this would take some more work, also we need to utilize some techniques to prevent overfitting further and to make the training process more efficient. We are currently studying some web tutorials and articles that show efficiency improvement using adaptive training rates and pruning techniques. We will further discuss our efforts in our final report and presentation.