Vehicle Recognition on Stanford Image-Set

By: Tyler Bryk

A popular application of machine learning is image processing. Image processing and classification is vital when it comes to autonomous vehicles, medical imaging, and security. Image processing is also used across many everyday uses such as image organization, social media, marketing campaigns, and photo enhancements. For this project, image classification will be used to identify the vehicular class of automobiles. This application of machine learning has the potential to be exceptionally useful for different roadway facilities such as toll plazas and parking garages, which charge different rates based on vehicular class. The project will consist of a convolutional neural network classifier which maps automobile images to one of 196 vehicle classes.

In terms of project data, Stanford's Cars Dataset is used, which contains over 16,000 images of cars and their respective labels which compromise 196 different features, based on the vehicles year, make, and model. The dataset is already balanced by class, meaning that there are an equal number of samples per each of the 196 classes. Additionally, the data has already been split into training and testing sets following a 1:1 ratio (8K training and 8K testing data). As far as data transformation is concerned, the images were resized to a uniform 256x256 size, and the RGB pixel values were normalized using PyTorch's built-in feature normalizer. This function finds the average pixel value across all images in a set, and then subtracts that calculated average from the training images.

The Densely-Connected Convolutional Neural Network (DenseNet) was chosen as the classifier for this project. As far as the implementation process goes, the data was loaded in from cloud files and transformed using the image resize and normalization parameters mentioned above. The DenseNet model was also loaded from PyTorch, and the architecture was configured so that the model would have 196 outputs. Additionally, the PyTorch optimizer, lr-scheduler, and cross-entropy loss function was used. The model parameters initially started out with a learning rate of 0.01 and momentum of 0.90. Due to the size and complexity of this DenseNet model, a GPU accelerator could not be used because it would simply overflow the CUDA memory. Instead, a standard CPU processor was used to train the model which took around 8 hours. Once tested, the in-sample accuracy was 99.840%, and the out-of-sample accuracy was 78.845%.

The parameter tuning process for this model was slightly less extensive due to the time complexity of the model. Being that 8 hours were required to train the model after each change in parameter, only a limited number of parameters were tested, which included learning rate and momentum. The DenseNet model comes in a few different flavors: 121-layer, 161-layer, 169-layer, and 201-layer architectures. The DenseNet also comes with the option of being pre-trained. For the model architecture, the 121-layer version was initially chosen, and that ultimately ended up performing the best. The higher-layer models were trained, the 161 and 169-layer models yielded accuracies that were near identical to that of the 121-layer, but the model complexity was large, and the training time was very costly. When testing the models as pre-trained, having that flag set to 'true' yielded nearly a 6% testing accuracy boost in all cases, so that flag was left on for the remainder of the tuning process. The number of iterations was also tested, but with a smaller range, and the optimal number was determined to be 10 iterations.

Overall, the DenseNet model performed very well for this application of image processing. The final model accuracy was nearly 80% which is very competitive to some of the state-of-the-art algorithms being tested by big machine learning researchers. The project offered further insight into Python coding, and also allowed for advanced library use including Pandas, Numpy, and PyTorch.