Capstone 2 Milestone Report 1

This project is intended to build a personalized model which is able to distinguish between a set of situations given an image. In this case the model will be trained on determining if a dog is in or outside of its crate, however, this idea can easily be generalized to monitor and alert users for a number of different situations.

Although the model itself would only be capable of classifying these personal situations based upon an image it could easily integrated into a system which monitors using a video camera and alerts a user based upon different classification readings. For instance, a sequence of situations could trigger an email or text message to a user which could provide valuable real time information making it a useful product for some consumers. The heart of this system would be the classification model engineered in this project.

A system such as this is useful for situations where simple electronic triggers are not applicable or too expensive to be useful in classifying what is going on. Going back to the example used in this project, a dog crate with electronic features is unnecessarily expensive and would arguably still lack the robustness of a machine learning monitoring system. Other useful situations could be monitoring a baby in a crib or monitoring places such as kitchen cabinets or a garage where you want to be alerted of certain people entering. These problems are simple to a human but have no easy automated solutions without the use of machine learning.

To develop the model, the system will use a set of videos labeled as situations that need to be recognized. In this case it will be the dog in the crate, the dog not in the crate, and the dog being taken out of the crate. The videos will be parsed into a set of still images to train the model. The model will use Deep Neural Networks for this classification process.

For generating the data, six separate videos were taken, two for each class in the problem averaging around a minute and a half each. The videos were parsed into JPG files for each frame using a ‘Free Video to JPG Converter’ application from Microsoft. Then using the Python module ‘skimage’ the images were iteratively resized, reshaped, classified, and stored

into a set of CSV files. The images were also converted from color into grey scale. This was done because color images take up three times the amount of memory and caused performance issues in the project. Each image was rescaled to 8% its original size, again this was tuned for optimizing both image quality and system performance.

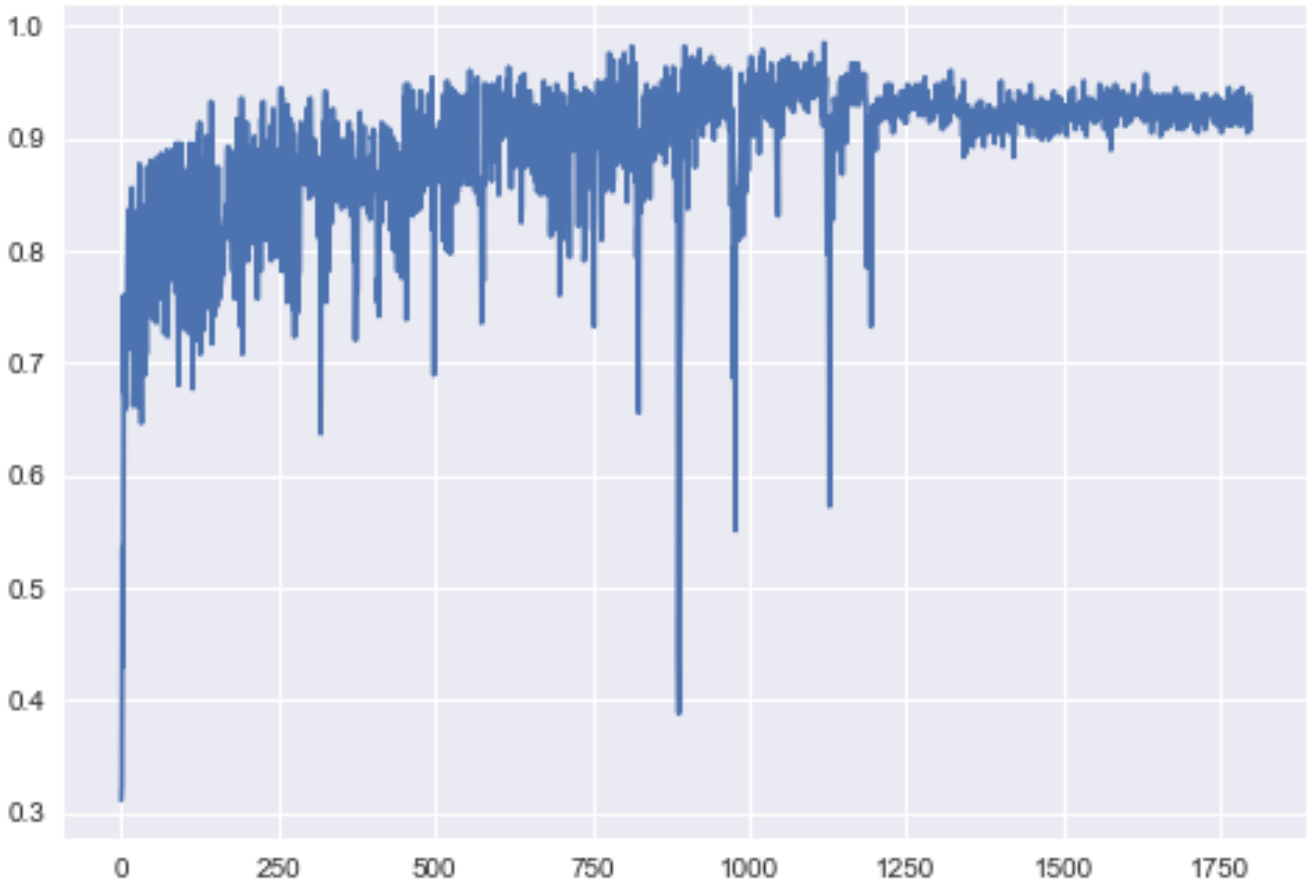
After resizing the images, they were flattened into a single dimension, making them convenient for storing as data tables and CSV files. Upon use, these images can be reshaped into their original image dimensions in order to take advantage of local correlations. Videos were taken with only one class occurring through each one, and because of this by knowing the video the image came from classifying each image was automated. Finally, sets of 50 cleaned images were saved to a CSV and given a randomized name. The randomized names of the file help to mix up the classes of data so that the model is trained on all class types for each batch. The data was spread into multiple files due to RAM limitations.

Overall, over 10,000 images were generated each consisting of over 5,000 pixels occupying nearly 7 GB of memory. This created issue in the project when attempting to push the data to Github. The large size of the files prevented all of the data from moving to the remote repository and because of this data was stored locally with only examples of the images and CSV files stored on the master branch.

Before using neural network models, a benchmark model was developed for reference using a more standard algorithm. A Random Forest Classifier was chosen because it integrates easily with multiclassification problems. In order to prevent the model from heavily overfitting the data was compressed to 10 features using a Principle Component Analysis algorithm implemented in the Scikit-Learn library. After the resizing the data and fitting to the Random Forest Classifier (also from Scikit-Learn) the model had an impressive 88% accuracy. This could most likely be increased through hyper-parameter tuning and experimenting with the algorithm or data feature size, but a general measure of the performance of more simple models was all that was needed.

After the benchmark model was create a simple dense neural network model was generated and fit to the data, using the Keras high level API. At first the following process was

used, iterate through each training file and fit the data to the model (meaning a Stochastic Gradient Descent update was made). Every 10 updates test the model on a reserved set of testing data. After each file had been fit on the model an epoch was complete. The first model was trained on 300 epochs and the performance was tracked. The peak performance of the model was over 98% accuracy, far greater than the benchmark model that was created. However, the performance was inconsistent over epochs varying as low as 80% accuracy at times.



The proposed solution was to further randomize the files into larger more integrated sets of data. In the original data collection process entire videos comprised of only one category were parsed in small datasets of again one category of very similar images. The idea was that but stochastically updating the model with these uneven datasets could cause the model to overfit and move into a lower performance configuration. To hopefully fix this issue the datasets were aggregated for every 10 small files into 1 larger csv file. By randomly aggregating

and shuffling the data each batch fit for the model would have a more even distribution of class labels.

After rearranging the data, a new set of neural network models were created ranging in the size and number of layers. The models were trained over span of 150 epochs in a very similar method as was done with the first neural network model. The models had a peak accuracy just of 99% which was very impressive. Unfortunately, all the models performed relatively equally in the long run and the consistency of the models was not increased by evening the label distributions of the data (as described in the previous chapter).