Programming Challenge Report

Thomas Berkane

1 Approach

Two approaches were considered to address the challenge: deep learning, and manually engineering features from the image data which would then be fed into a machine learning model to predict the class. While the first approach would likely give greater accuracy, it is also more complex and much slower. So since speed is an important requirement of the challenge, the second approach was chosen. All of the solution code is contained in a Jupyter Notebook which can be found at https://github.com/tberkane/biped-challenge.

2 Features

In the "Building the dataframe" section of the notebook, the following features are generated for each image pair:

- **avg_brightness**: the mean of every pixel value in the grayscale version of the black and white image. This is to distinguish between the "low-light" and "bright-light" classes. I saw that other metrics for quantifying illumination exist, such as perceived brightness, which might be useful in further work to improve this feature.
- **bright_pixels**: the number of pixels in the grayscale image whose value is greater than a threshold (220). This feature is useful for identifying strong light sources or reflective surfaces in the images and thus to distinguish the "bright-light" class.
- **near_pixels**: the proportion of valid (not 0.0) pixels in the depth image whose depth is less than a threshold (1). This is to help identify images which have objects covering the camera.
- **depth_sd**: the standard deviation of all pixel depths in the depth image, excluding invalid values. The idea was to help identify the "covered" class because since there are two 'layers' in those images, I thought that the depth would have greater variance. It turns out that the opposite is actually true because "covered" images have the lowest mean standard deviation of depth of all classes, but this feature seems to help the model so I kept it.
- **black_pixels**: the number of invalid values in the depth image. This feature was added to help distinguish the "covered" class because I noticed that many of the objects which cover the camera have invalid depth values.

3 Models

For the classification of the image pairs, in the "Model training and prediction" section of the notebook, we try three different models: multi-class Logistic Regression, Support Vector

Model	Accuracy	Tuned hyperparameters
Baseline	0.430	
Logistic Regression	0.733	C, multi_class
SVM	0.633	C, penalty
Random Forest	**0.900**	min_samples_leaf, min_samples_split, n_estimators

Table 1: Best obtained accuracy for the different models with hyperparameters tuned.

Machines, and Random Forest. Each model is trained on 80% of the images and then tested on the remaining 20%. To tune the hyperparameters, I apply 5-fold Grid Search Cross Validation.

To compare the models, I use accuracy as a metric. The baseline accuracy is computed by considering a model which always predicts the most frequent class ("normal" in our case), which gives an accuracy of about 0.430.

Table 1 shows the results for the different models. We can observe that the model which performs best, with an accuracy of 0.900, is Random Forest. The optimal hyperparameters are min_samples_leaf=1, min_samples_split=2 and n_estimators=1000.

4 Visualization Tool

The "Simple visualization tool" section of the notebook provides a visualization of the extracted features for each image, as well as the predictions of the best model. Please see the notebook for more details.

5 Benchmarking

To evaluate the running time of the found model, I use the IPython magic function %timeit in the "Benchmarking" section of the notebook and measure two metrics: the time to train the model on all images except one, and the time for the model to output a prediction for the image which was left out.

The training takes 963 $\pm$ 60.2 ms, and the prediction takes 72.8 $\pm$ 0.987 ms. This means that we can make predictions for about 13.7 images per second.

6 Further Work

Although the best model's performance is quite good relative to the baseline, there is room for improvement. First, we could look at the model's confusion matrix to see which classes are being wrongly classified as others, and thus figure out what class of images to focus on to find better features to extract. For instance, even though I have a feature for the number of bright pixels, low-light images still have quite a large number of those because of the white dots from the infrared sensor. So it might be worth spending time to find a more complex feature to distinguish low-light situations. Also, it could be useful to perform an ablation analysis to figure out which features are actually helping. Finally, given how well the Random Forest model performs, we could try out other ensemble models such as Gradient Boosting.