Problem description

Our goal was to create a convolutional neural network for recognizing different classes of hand gestures in real time in order to simulate a car game controller.

We decided to create two separate models. We used one of the models to detect gestures for turning a virtual steering wheel left, forward and right, we will refer to this model as the “wheel model”. We used the other model for detecting gestures for “stepping” on the “gas pedal” and releasing it, as well as breaking and going in reverse, we will refer to this model as the “pedal model”.

We tested using binarization, thresholding and Canny edge detection as preprocessing algorithms, however we found that the data was too diverse for using a preprocessing algorithm which would work in most of the cases.

We used Bayesian optimization to optimize a set of hyperparameters which we used to train our final models.

Models

The “wheel model” was trained to detect three classes, one for each state of the wheel (left, forward and right). The “pedal model” was also trained to detect three classes, one for stepping on the gas pedal, one for releasing the gas pedal and one for breaking and going in reverse.

It was important for us to create small enough architectures so that they could run concurrently along with the game which they were used for.

Dataset

For our dataset we created an automated data acquirement tool which could be configured to record the different hand gestures for both models for training, validating and testing.

For the wheel model we recorded around 6000 images per class and we used those images for training. We recorded 800 images per class that we used as the validation data during training. During training we would test out our model in games and when it underperformed, we recorded validation data with the same gestures we would use in-game so that we could continue training accordingly. After training we recorded another 850 images per class for calculating the confusion matrix and mean per class accuracies. We used the same approach for the pedal model.

Model	Train per class	Validation per class	Test per class
Wheel model	~6000	~800	~850
Pedal model	~5200	~580	~430

Experiments

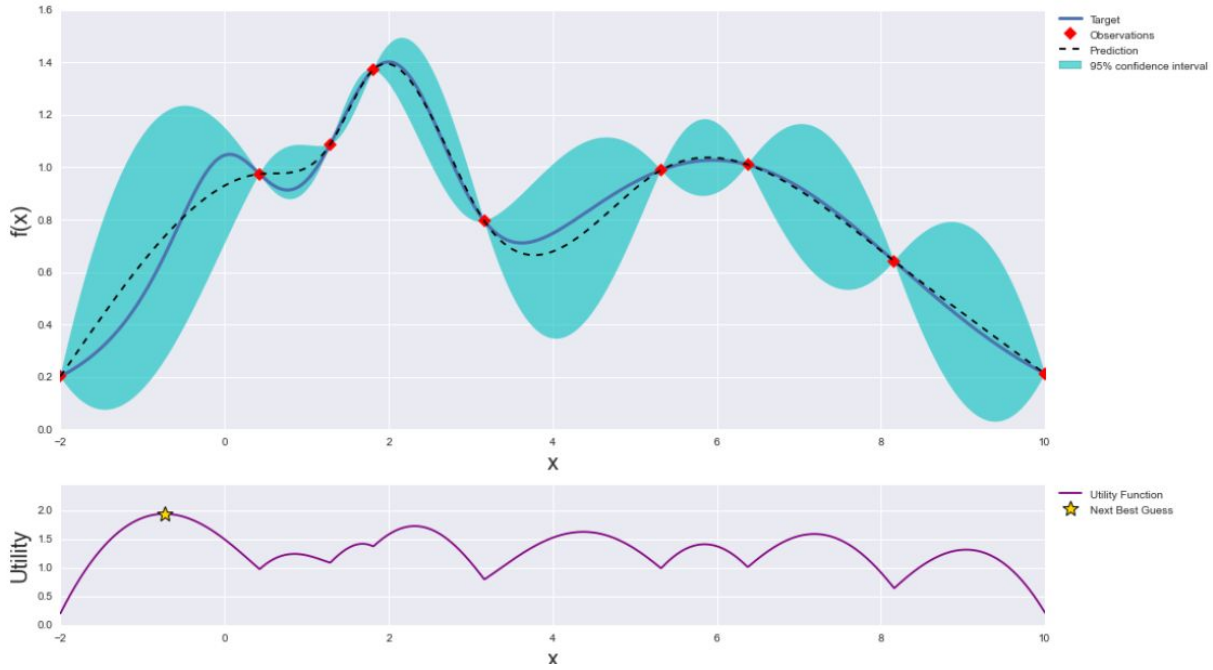
We used keras with a tensorflow backend to create, train, test and use our models. We created a couple of different models with fixed hyperparameters to create a proof of concept. Each set of hyperparameters were trained for 200 epochs. The duration of training for 1 set was around 1.3 hours.

The process of manually trying out different hyper parameters was too slow so we started thinking about automating the process somehow. The first idea was to create a parameterized model so that the hyper parameters could be defined separately, and we would create lists of hyper parameters and run an automated training session in which all sets of predefined hyper parameters would be tested and the results recorded along with the weights for each set.

We created the models and decided to use Bayesian optimization instead of manually defined hyper parameter sets.

“Bayesian optimization works by constructing a posterior distribution of functions (gaussian processes) that best describe the black box function we want to optimize. As the number of observations (validation accuracies of the training attempts) grows, the posterior distribution improves, and the algorithm becomes more certain of which regions in the hyper parameter space are worth exploring and which are not, as seen in the picture below”.
- [partial quote: https://github.com/fmfn/BayesianOptimization]

The larger blue areas denote the amount of uncertainty in the given region.



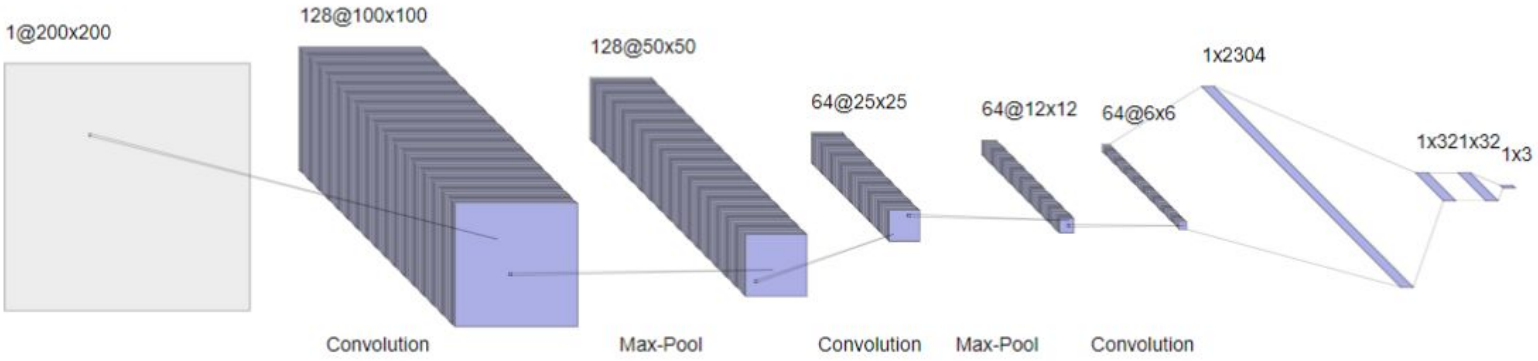
[image source: https://github.com/fmfn/BayesianOptimization]

We ran the Bayesian optimization process for ~30 hours. The process involved training on 23 different sets of hyper parameters for 200 epochs, and the best set among them had a validation accuracy of 0.89 and was also relatively small. We decided to train that set further.

To improve the validation accuracy we recorded more training data, used data augmentation and a learning rate reducer, which reduced the learning rate when it detected that the validation accuracy is plateaueing.

Using these methods we achieved 0.95 validation accuracy for the wheel model and 0.93 validation accuracy for the pedal model.

The final architecture for both models is displayed in the image below. Note that batch normalization layers, relu activations and dropouts are not displayed. There is a batch normalization layer before each relu activation, and a relu activation after each convolutional and fully connected layer.



Bayesian optimization results

Filter sizes and strides were not optimized we used 2x2 for both. The best set of hyper parameters that we found is marked with a red border.

Set #	Conv1 filters	Conv2 filters	Conv3 filters	Fully Connected 1 Units	Dropout after FCL1	Fully Connected 2 Units	Dropout after FCL2	Validation accuracy
1.	112	75	38	113	0.36	56	0.325	0.742
2.	124	79	38	135	0.294	138	0.407	0.783
3.	100	100	64	32	0.5	150	0.499	0.816
4.	128	64	32	32	0.499	150	0.5	0.768
5.	100	66	64	122	0.2	150	0.2	0.881
6.	100	64	32	150	0.5	150	0.5	0.76
7.	128	64	64	32	0.2	32	0.2	0.894
8.	128	64	64	150	0.5	32	0.499	0.74
9.	100	64	64	32	0.2	80	0.482	0.747
10.	128	100	64	32	0.5	32	0.2	0.783
11.	100	100	64	150	0.2	150	0.2	0.885
12.	128	64	32	32	0.2	32	0.2	0.752
13.	128	100	64	81	0.5	150	0.2	0.815
14.	128	64	64	150	0.2	150	0.2	0.83
15.	100	64	64	76	0.2	32	0.2	0.84
16.	128	64	64	77	0.2	32	0.499	0.765
17.	100	100	64	150	0.2	32	0.2	0.794
18.	100	64	64	150	0.2	92	0.2	0.820
19.	128	64	64	32	0.2	150	0.2	0.798
20.	100	64	64	32	0.5	32	0.49	0.744
21.	100	64	32	150	0.2	32	0.2	0.798
22.	100	100	32	90	0.2	150	0.2	0.847
23.	100	100	64	102	0.2	111	0.2	0.843

Results

Per class mean accuracy for the wheel model:

Left	Forward	Right
0.997	0.891	0.988

	Left	Forward	Right
Left	3276	8	0
Forward	358	2935	0
Right	34	4	3289

Per class mean accuracy and confusion matrix for the pedal model:

Pedal down	Pedal up	Break / Reverse
0.997	0.852	0.949

	Pedal down	Pedal up	Break / Reverse
Pedal down	3225	0	8
Pedal up	478	2755	0
Break / Reverse	162	0	3043

Conclusion

We have found that Bayesian optimization is a great optimization tool that can be used in practice for automating the process of finding the best hyper parameters for a given classification problem.

The data in our dataset was captured in two different positions and 5 people are present in the data. We would need more diverse data in order to better generalize our model.

Static preprocessing didn’t work out well because of the existing and potential diversity of the data.

We have found that even small CNN models can be used for emulating these kinds of controllers as long as the users of the controller capture their specific hand gestures and train the model so that it can better understand them and their environment.