GOLF-BALL VELOCITY CALCULATION THROUGH RADAR SPECTROGRAPHY

Filip Roszkowski (s233421), Marcel Zelent (s233422), Karl Mihkel Seenmaa (s232512)

Technical University of Denmark
GitHub

ABSTRACT

This project focused on optimizing a Machine Learning (ML) model developed by Trackman to predict the velocity of a golf ball at the time of contact with the golf club. The basline model contains 38M parameters and uses convolutional and linear layers. The team aimed to reduce the model parameters while maintaining or improving the model's performance. Four approaches were explored: parameter tuning, distilled knowledge, LSTM, and quantization. The first three approaches aim to reduce the size of the model, while quantization aims to minimize computational demands during inference. The team came up with a variation of different models and techniques that downsize the model and increase the inference speed. In the end, the team was able to achieve a much smaller and faster model.

1. INTRODUCTION

Machine Learning (ML) models are often characterized by their large parameter count, enabling high accuracy with complex datasets but introduce a problem of slow predictions in resource constrained real-time systems. This project aims to optimize the size of a Trackman ML model, which predicts the velocity of the golf ball from spectrogram data obtained from four radar receivers. This information is then used as input to a high-fidelity simulator of a golf-course.

The measurements, derived from four radar receivers, include power spectrograms for each receiver and phase spectrograms, resulting in six input channels to the ML model.

The Trackman team has already found a suitable model with satisfactory performance for this task. However, since this is a commercial product, the ML model runs on an embedded device with limited memory and performance, thus warranting the need for optimization with regards to size. Therefore the goal of the project is to find a method of downsizing it, while still keeping an error value the same or lower than the original. Furthermore, it is desirable if the smaller model can also operate with faster prediction speed.

The baseline model consists of 38M parameters, with an architecture consisting of 4 convolutional layers (with max-pooling and ReLU activations) and three linear layers. These will be from now on referred to as *CNN* and *ANN*. The aim of the team was to significantly reduce the model's size and computational complexity while maintaining or improving its accuracy and inference speed. Thus, the following four approaches were explored to achieve this goal:

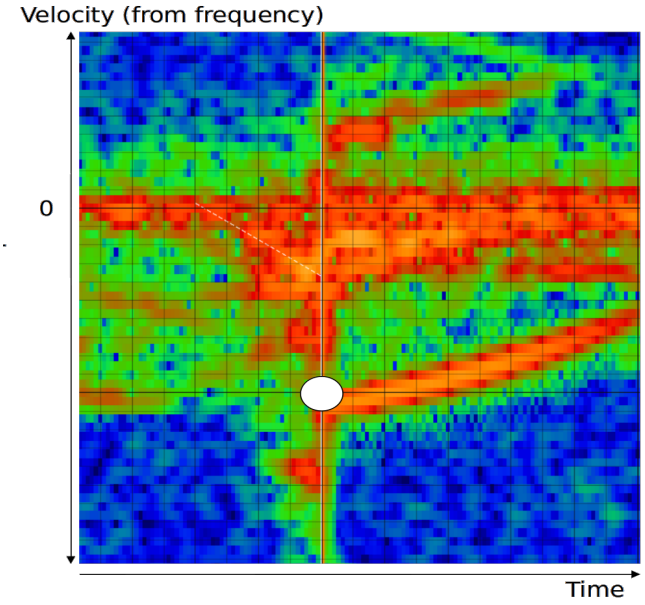


Fig. 1. Spectogram containing velocity from frequency data.

- **Parameter tuning**: Systematic reduction of network parameters.
- **Long Short-Term Memory (LSTM) networks**: Using LSTM networks to handle sequential data derived from radar inputs.
- **Distilled Knowledge**: Employing the teacher-student paradigm.
- **Quantization**: Converting high-precision floating-point operations to lower-precision integer operations for faster inference.

2. PARAMETER TUNING

During the initial examination and analysis of the baseline model architecture, the team realized that the main part of the model's parameters comes from the ANN block (3 linear layers). More precisely, ca. 99% of the weights come from the first linear layer, where the network downsizes from 37120 inputs to 1024 outputs. Thus, in the following development the group kept in mind the fact that in order to downsize the model parameters significantly, the inputs to the ANN block had to be way smaller.

To find the optimal sizes of the CNN and ANN, each of these were consequently downsized and checked if the prediction accuracy had stayed roughly the same. In case of the CNN, it was possible to shrink it down by a factor of 2 - at that point the RMSE had an increase of 6.25% above baseline. In the same way, it was possible to shrink the ANN by a factor of 4 (apart from input size, which stayed the same). The RMSE during training for both models and the baseline can be seen on Figure 2.

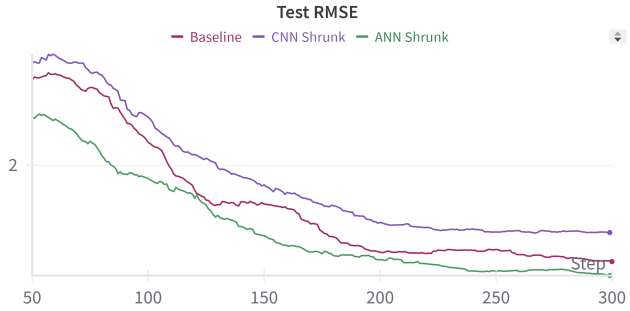


Fig. 2. Plot of the RMSE from the Test dataset. Certain transformations were used for better clarity.

The approach for developing a combined model was simple - the input parameters to the ANN block had to be reduced. There were multiple options to achieve that - max-pooling, reducing channel sizes, and adding more or modifying convolutional layers and their kernels. When experimenting around with the different options, we found that the addition of more convolutional layers proved to reduce the loss of the model, while also reducing the output size from the CNN block. Thus we decided to add 5^{th} convolutional layer to the model, that would further downsize the parameters, and enhance the learning of more complex feature-maps. The ANN layer input size was simply matched with output size of the CNN block and rectified linear unit activation functions with dropouts were added after the first and second linear layers of the ANN block. As a result the final combined model was downsized ca. 9 times, and reduced the test RMSE from 1.59 m/s to 1.29 m/s as can be seen in Table 1.

	Test RMSE	**Test Log RMSE**	**Model parameters**	**Size [MB]**
Baseline	1.59	0.20	38M	153.67
CNN shrunk	1.69	0.23	20M	73.6
ANN shrunk	1.54	0.19	10M	36.8
Combined	1.29	0.11	4.5M	17.35

Table 1. Parameter tuning model metrics.

3. LSTM

LSTM networks are well-suited for handling sequential data, making them a good candidate for this project. LSTMs are good a capturing temporal dependencies with their ability to have memory and by leveraging memory gates, which allows them to disregard or add more emphasis to information over time. This time however, our input data to the networks weren't spectrograms, but sequences of radar recording from the 4 different radars.

To implement this approach, the input data was organized into temporal sequences using a custom data loader. These sequences were constructed from the radar receivers, aiming to capture temporal relationships between consecutive spectrogram frames. Despite our concept being in line with the sequential structure of the data, the LSTM models did not perform as well as hoped. We discovered several challenges, that could explain the reason for that:

- **Noise in the data**: The raw sequential data from receivers included noise, which may have played a role in the LSTM's ability to effectively learn meaningful patterns.
- **Memory limitations**: Training LSTMs with a large number of parameters requires significant computational resources, which the group did not have. To address this, the LSTM models were downsized, but this reduction in capacity meant that we may have limited their ability to generalize on test data.
- **Model architecture**: The LSTM architectures implemented may not have been optimally designed for this specific use case. However, further experimentation with hyperparameters and network structures could lead into better results.

4. KNOWLEDGE DISTILLATION

While researching various methods for optimizing the size of a model, the team has stumbled upon a technique known as **Knowledge Distillation**. First introduced by Hinton et al. in their influential paper, "Distilling the Knowledge in a Neural Network" [1], the method has seen plenty of use in both academia and industry, specifically due to how useful it is in

compressing very large models. While usually the goal is to optimize the size to accommodate scaling the platforms to millions of users, it is fundamentally the same issue as fitting it on a small hardware platform like it is in case of Trackman.

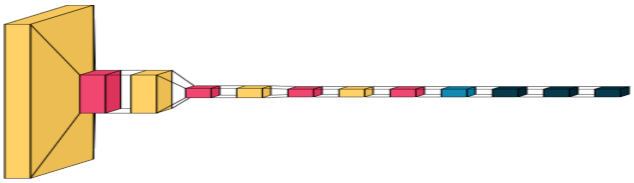
There are many ways to implement *Knowledge Distillation*. Gou et al., in their comprehensive survey "Knowledge Distillation: A Survey" [2], categorize various distillation techniques based on the types of knowledge transferred (e.g., logits, feature maps, or attention mechanisms) and their training methodologies. They also emphasize that knowledge distillation is not limited to model compression but extends to tasks such as domain adaptation, robustness against noisy labels, and improving interpretability.

The specific method that was used by the team is on the simpler side. The basic concept behind it is to transfer the knowledge from a large, complex model (referred to as the "teacher") to a smaller, more efficient model (the "student"). The knowledge transfer was achieved by modifying the *loss function* of the student.

$$L_{total} = L_{hard} * \alpha + L_{soft} * (1 - \alpha) \quad (1)$$

The hard loss is the loss from normal training, while soft loss is defined as the difference between the prediction made by the teacher and the student. Furthermore, the α parameter can be used to adjust the ratio between them.

The team thus took the baseline model as the teacher, and then made an appropriately small student model, using the experience gained during parameter tuning. As a result, the team was able to achieve a model with only **6,651 parameters**. The layers in both the CNN and ANN were minimized as much as possible, which is shown on Figure 4. An important step here was to increase the kernel size of the pooling layers, which meant that the number of input neurons to ANN was down to meager 55 (37120 originally). As a result of all of these adjustments, the final size of the model was **32 KB**.



The method does have a negative effect on the performance, as now the Test RMSE was around 1.54 in comparison to 1.29 of the *Combined Simplified* model. Nonetheless, that is within the range of the original baseline.

At this stage, the team decided that any further improvements would not bring significant gains towards the ultimate goal. Nonetheless, it can be of interest to explore other methods of distilling knowledge. Recent advancements in the field have expanded the scope and understanding of this approach. Tang et al.'s work "Understanding and Improving Knowledge Distillation" [3] sheds light on the mechanisms behind distillation and suggests ways to enhance its effectiveness through better teacher-student alignment and refined loss functions. Furthermore, Zhang et al.'s study, "Quantifying the Knowledge in a DNN" [4], offers a theoretical perspective, quantifying the knowledge captured by neural networks and explaining how distillation preserves critical information while reducing model size. Any further work would be most likely comprised of analysing these papers and implementing them into the problem.

	Layers	Channels	Notes
CNN	4	Baseline: 6-16-32-64-128 Student: 6-7-8-9-11	Baseline Pooling Kernel = 2 Student: Pooling Kernel = 4
ANN	3	Baseline: 37710-1024-256-1 Student 55-24-5-1	Added ReLU activations

Table 2. Student network architecture

5. QUANTIZATION

Quantization is a process in which the 32-bit floating point numbers used inside the model are turned into 8-bit integers (16-bit floats for GPU). This considerably saves on inference time, as the Arithmetic Logic Units (ALU's) inside a modern chip are more optimized to work with these numbers. The process starts by teaching a Machine Learning model to a high degree using floating point numbers, and then transforming it into the simplified version by a process called calibration. During calibration, inference is preformed on the model while the quantization framework observes the minimum and maximum values of the each neuron input in order to properly rescale the weights and biases to fit the expected range of inputs - so that the impact of the lower precision is minimized.

There are various way to quantize a model:

- dynamic (runtime) quantization
  - CUDA - TensorRT [5]
- static (post training) quantization
  - x86 - Fbgemm [6]
  - ARM - Qnnpack [7]
- quantization aware training

Presented on this paper, are the Static and Dynamic methods. As can be seen on Figure 3 and Figure 5 where the static

quantization results are shown for x86 and ARM platforms. This is the main method used for CPU inference speed-up. The quantization is done after the model training and is done only once before the inference. As for the GPU as seen on Figure 4, Dynamic quantization is preferred. It is done just-in-time before the inference and during the runtime, this way it has an added benefit of being able to adjust the calibration to unseen data were as the static quantization can not be changed online. All of the quantized models preformed within 0.01% of RMSE of the non-quantized models. The model file size difference can be seen in Table 3.

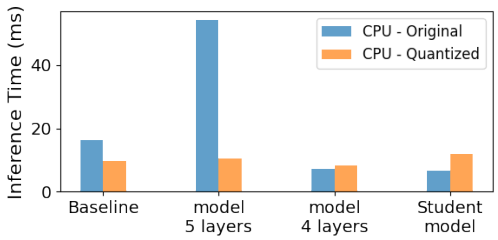


Fig. 3. Inference Comparison (CPU, Intel® Core™ i7 10th Gen @ 4Ghz)

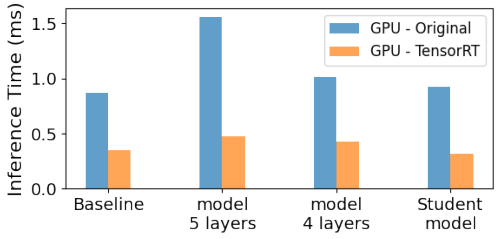


Fig. 4. Inference Comparison (GPU, NVIDIA GTX 1650 Ti)

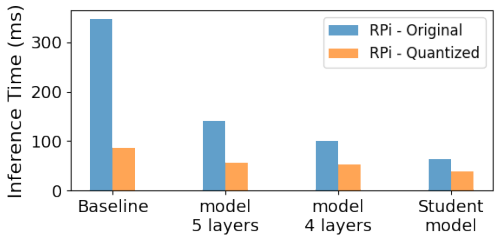


Fig. 5. Inference Comparison (Raspberry Pi 4, Broadcom BCM2711 @ 1.8Ghz)

	Baseline	**model 5 layers**	**model 4 layers**	**Student model**
Original size [MB]	153.67	17.35	19.22	0.03
Quantized size [MB]	38.46	4.39	4.83	0.02

Table 3. Model file size comparison.

6. CONCLUSION

In this project, we explored various methods to optimize a baseline Machine Learning (ML) model for predicting golf ball velocity from radar spectrogram data. The baseline model, while accurate, was too large and computationally demanding for embedded systems. By employing parameter tuning, LSTM networks, knowledge distillation, and quantization techniques, we achieved significant reductions in model size and inference time without compromising prediction accuracy.

Parameter tuning enabled us to reduce the number of model parameters by approximately 9 times, achieving better performance than the baseline. Although the LSTM approach showed promise for handling sequential radar data, its performance was limited due to noise, memory constraints, and suboptimal architecture. Knowledge distillation proved to be highly effective, allowing us to shrink the model down to just 6,651 parameters and a mere 32 KB in size, with an RMSE close to the baseline. Finally, quantization demonstrated substantial reductions in file size and inference time on both CPU and GPU platforms, making the model even more suitable for real-time embedded applications.

The optimized models not only meet the hardware constraints of Trackman's embedded devices but also maintain or exceed the accuracy of the original model. Future work could focus on improving the LSTM architecture, exploring advanced distillation techniques, and refining the quantization process further for specific hardware platforms.

7. REFERENCES

[1] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean, "Distilling the Knowledge in a Neural Network," `https://arxiv.org/abs/1503.02531`, 2015.

[2] Jianping Gou, Baosheng Yu, Stephen J. Maybank, and Dacheng Tao, "Knowledge Distillation: A Survey," *International Journal of Computer Vision*, vol. 129, no. 6, pp. 1789–1819, Mar. 2021.

[3] Jiaxi Tang, Rakesh Shivanna, Zhe Zhao, Dong Lin, Anima Singh, Ed H. Chi, and Sagar Jain, "Understanding and Improving Knowledge Distillation," `https://arxiv.org/abs/2002.03532`, 2021.

[4] Quanshi Zhang, Xu Cheng, Yilan Chen, and Zhefan Rao, "Quantifying the Knowledge in a DNN to Explain Knowledge Distillation for Classification," https://arxiv.org/abs/2208.08741, 2022.

[5] NVIDIA Corporation, "NVIDIA TensorRT: High-Performance Deep Learning Inference," https://developer.nvidia.com/tensorrt, 2024, Accessed: 2024-12-20.

[6] Khudia Daya, Huang Jianyu, Basu Protonu, Deng Summer, Liu Haixin, Park Jongsoo, and Smelyanskiy Mikhail, "FBGEMM: Enabling High-Performance Low-Precision Deep Learning Inference," *arXiv preprint arXiv:2101.05615*, 2021.

[7] Marat Dukhan, Yiming Wu, Hao Lu, and Bert Maher, "QNNPACK: Quantized Neural Networks PACKage," https://github.com/pytorch/pytorch/tree/main/aten/src/ATen/native/quantized/cpu/qnnpack, 2024, Accessed: 2024-12-20.