# An improived deep learning methodology for golf ball intial velocity estimation from doppler signal

Artur Habuda s233190
Xixi Wang s232253
Alexandru-Stefan Marin s232414
Vincenzo Genovese s232434

December 15, 2024

## 1 Introduction

TrackMan uses advanced radar and camera technology to capture precise data on golf club motion and ball trajectory, such as ball speed, spin rate and launch angle, to help athletes train. But the signal is confounded by noise arising from other moving components during the swing. Therefore, we need to estimate the initial radial velocity of the ball after the club hits it based on the spectrum converted from the time domain waveform of the Doppler radar.[1] To achieve these goals, we introduce skip connections, which improve the flow of information between layers and eliminate the gradient problem, an approach that allows the network to be efficiently trained without a significant increase in parameters and Global Average Pooling to decrease the number of parameters. Two different models are designed.

## 2 Project Goals

The primary focus of this project was to estimate the radial velocity of a golf ball using six-channel spectrogram data as input for a Convolutional Neural Network (CNN). The objective was to achieve higher accuracy in velocity estimation while minimizing the number of trainable parameters in the model, with respect to a Baseline model.

## 3 Data

The dataset consists of six-channel spectrogram data .**4 power channels**: these channels capture the scattering properties of the signal and are primarily used for ball detection. **2 phase channels**: these channels provide phase difference information, which is critical for velocity estimation.

These radial velocities can be interpreted from the Doppler shift observed by the radar system according to eq. (1):

$$v_r = f_d/(2 * f_0) * c \tag{1}$$

where $v_r$ is the radial velocity, $f_d$ is the Doppler shift, $f_0$ is the centre frequency of the transmitter, and $c$ is the speed of light.

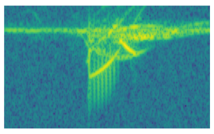
Figure 1: Power Spectrogram

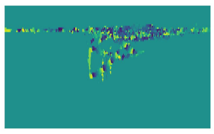
Figure 2: Phase Spectrogram

The spectrogram data available for this project was cropped beforehand removing non-informative areas. Thus the working data are extracted sub-spectrograms of time in [-150, +200]ms and velocity in [-60, + 15]m/s. These spectograms are saved in ".npy" format, which is later min-max normalized, channeld-order swapped and interpolated to a fixed size for input for the CNNs(height=79, width=918, channels=6).

The dataset was split into training (1700 samples, 77.9%), validation (83 samples, 3.8%), and test (400 samples, 18.3%) sets. The initial split proposed by Trackman had test and validation splits switched. Although a higher sample count on the validation set, would have helped a more informed fine-tuning of the hyperparameters, it was decided to keep the test set bigger, in order to obtain a more robust final evaluation of the model.

## 4 Implementation

With the project goals in mind and after several iterations of trial and error, two main models came to fruition Residual-ReLU Regressor(RU) and Squeeze-Excite Residual-SiLU Regressor(SU). Next, an ordered description of the design process of both is presented:

Aiming to decrease Baseline's (BL) parameters, Global Average Pooling(GAP) was introduced after the convolution layers. Instead of flattening each

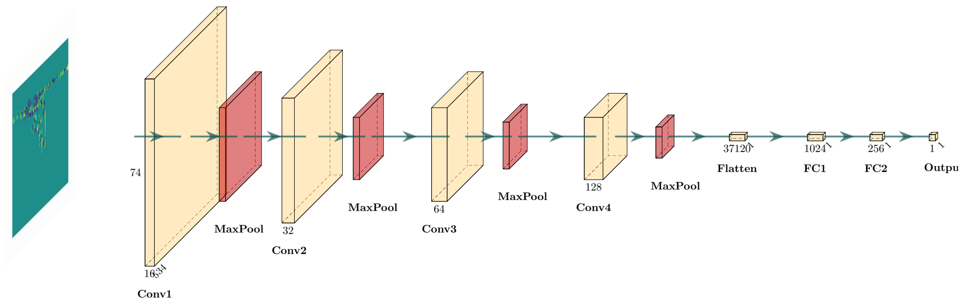

Figure 3: Baseline (BL) Architecture Diagram.

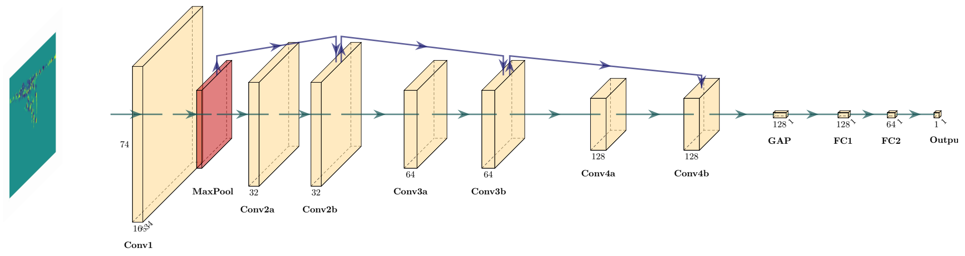

Figure 4: Residual-ReLU (RU) Architecture Diagram.

component of the feature maps and using them as input for each neuron in the final FCNN, GAP computes the average value of each feature map, effectively summarizing the information in that map into a single value. [2]. The objective of implementing GAP was two-fold; to reduce significantly the amount of parameters of the model, and to reduce overfitting (which was a common issue when training BL, Figs. 6, 7). In order to offset the aggressive reduction in dimensionality by GAP, while maximizing the retained information trough the network, MaxPool2D layers ere replaced with Convolutional Layers. The results were not ideal, therefore residual connection were incorporated, trying to include past information and reducing the risk of vanishing gradients. Additionally batch normalization was added in order to enhance convergence speed and training stability. Although the RU model was not very deep, leackyReLU was implemented in order to try to stabilize training. LeackyReLU allows a small, non-zero gradient for negative inputs; this allows the model to address the dying neuron problem and stabilize training, as more neurons contribute to learning. Nonetheless it is expected to extend the training time. Dropout was also included as a regularization technique. Other configurations that were tried on top of this model architecture, but which did not present improvements in testing were: an Adam optimizer, Kaiming-weights initialization, adaptive learning rate trough a scheduler and gradient clipping. Specifics on the impact of some of these parameters are show in table 2.

Training and prediction tests (discussed in Section 5) highlighted several challenges, including prolonged training and inference times, as well as convergence instability. To address these issues, a third model was devised, the Squeeze-Excite Residual-SiLU (SU).

Squeeze-and-Excitation (SE) blocks were introduced to enhance the feature extraction process by recalibrating channel-wise feature importance. The SE mechanism performs two key operations: **Squeeze**, where global average pooling compresses spatial dimensions to summarize each feature map into a single representative value, and **Excitation**, where adaptive scaling factors are learned and applied to each channel, amplifying informative features and suppressing irrelevant ones. This was achieved by creating a SE block that consists of Global Average Pooling (GAP) followed by two fully connected layers with LeakyReLU and Sigmoid activations, dynamically scaling feature maps to emphasize informative channels and suppress less relevant ones. This channel-wise attention mechanism should improve the model's ability to focus on critical patterns in the input data, leading to better feature representation.

The SU model also replaced the LeakyReLU activation function with SiLU (Swish), which is known for providing smoother gradients, enhancing training stability, and improving convergence. By allowing smoother transitions between activation states, SiLU reduces abrupt gradient updates, contributing to faster and more reliable training.

Each subsequent model aimed to address the limitations of its predecessor by incorporating architectural innovations and fine-tuning hyperparameters.

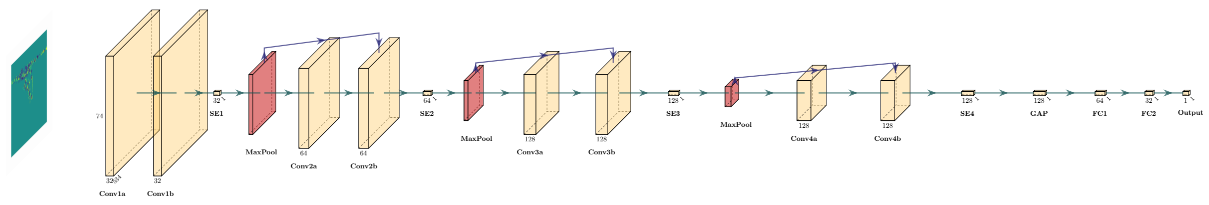

Figure 5: Squeeze-Excite Residual-SiLU (SU) Architecture Diagram.

## 4.1 Summary of Hyperparameters

Table 1 highlights the key architectural features, hyperparameters, and performance metrics for all three models.

| **Hyperparameters** | **Baseline (BL)** | **Residual-ReLU (RU)** | **Squeeze-Excite-SiLU (SU)** |
|---|---|---|---|
| Activation Function | ReLU | LeakyReLU (slope= -0.01) | SiLU |
| Optimizer | SGD | SGD | SGD |
| Learning Rate | $10^{-5}$ | $10^{-5}$ | $10^{-5}$ |
| Batch Size | 10 | 10 | 10 |
| Dropout Rate | None | 0.4 | 0.4 |

Table 1: Comparison of hyperparameters for BL, RU, and SU models.

## 4.2 Challenges and Lessons Learned

Despite the improvements, the RU model introduced computational overhead due to the sequences of computations, but yet is the better performing model. While the addition in the number of parameter for SU and the more advanced technique adopted in implementing the architecture, it could not overcome the results in error of RU. However, the training time for SU is lower than RU.

# 5 Results and Evaluation

All models rapidly decrease in training loss early on. After that BL's training loss continues to decrease steadily trough 500 epochs, achieving a notably lower RMSE than the other two models. RU and SU maintain a relatively higher training loss to baseline, with a slight edge for SU. Convergence

The picture changes in the validation phase. The curves in this phase are significantly more volatile, particularly early in the training. This suggests difficulties in generalization or sensitivity to certain samples/batches. The baseline model demonstrates relatively lower and more stable validation losses, eventually plateauing at a consistent level. Despite incorporating residual connections and batch normalization, RU's validation loss shows the most unstable behavior. SU, although still unstable, shows slightly more invariable validation loss than SU. It seems that incorporating SILU and and the Squeeze-Excite blocks had the expected effect, as explained in 4. The training RMSE plots mirrors the loss behavior in Fig 8.

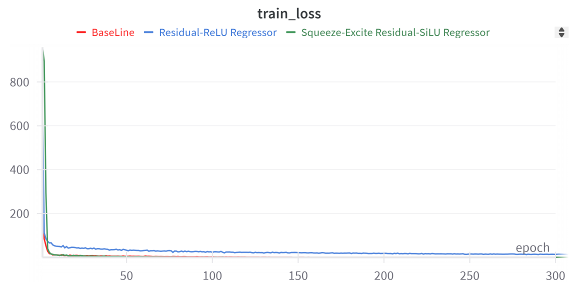

Figure 6: Train Loss in logarithmic scale of BA, RU and SU.

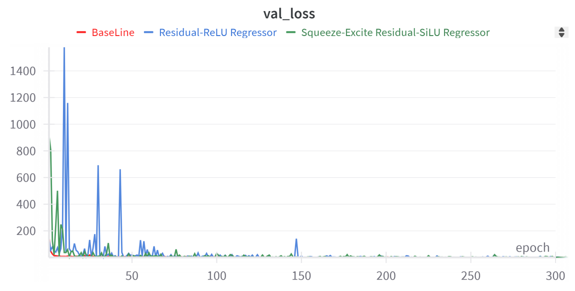

Figure 7: Validation Loss in logarithmic scale of BA, RU and SU.

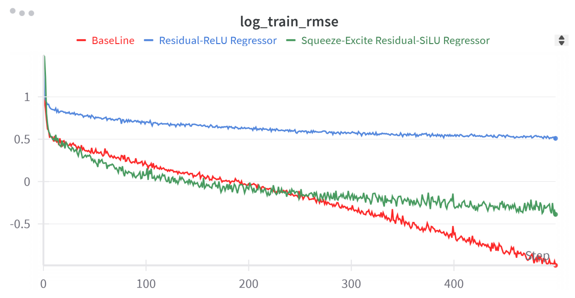

Figure 8: Train RMSE in logarithmic scale of BA, RU and SU.

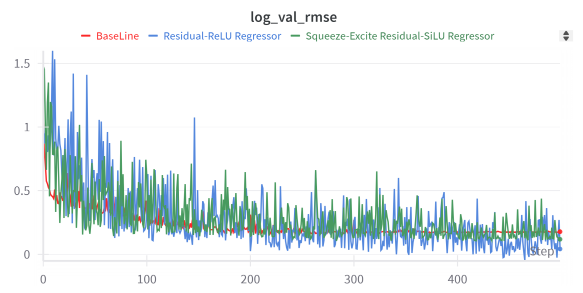

Figure 9: Validation RMSE in logarithmic scale of BA, RU and SU.

Note that validation convergence is reached no further than at around epoch 200 for any of the models. In order to maximize clarity, loss results are shown only until 300 epochs, and rmse outcomes are shown in log scale. Thus, both sets of graphs showcase best, the variability in results for all models, at the key stages of training and validation.

In Figure. 10, the results of the three models are shown.

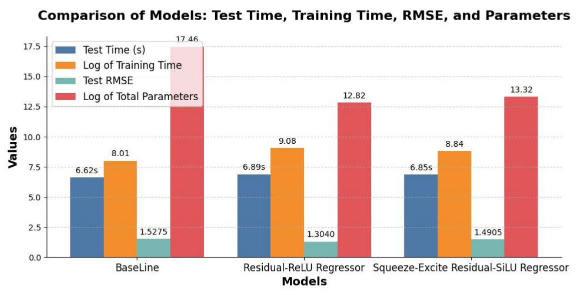

Figure 10: Final results outcomes of chosen three models: test RMSE, log. of training time, inference time and log. of parameter count.

In table 2 the impact of certain changes in the hyperparameters of RU, on the test RMSE values is shown.

| | **Adam** | **Kaiming** | **SILU** |
|---|---|---|---|
| Test RMSE | +41.89% | +6.64% | +7.99% |

Table 2: Impact of different hyperparameters(Adam optimizer, Kaiming Weights Initialization and SiLU activation function) on TEST RMSE of RU.

It shall be noted that these hyperparameter changes are specific to the RU architecture; other architectures may be affected differently. For example, SU saw a +1.54% increase in test RMSE when using leackyReLU instead of SILU.

RU provides the best test RMSE at the lowest parameter count. On the other hand, SU provides a good compromise between good performance and improved inference and training time over RU. As defined by Trackman, RU is the model that performs best. Nonetheless, it has been decided to keep SU in this evaluation, as it validates some of the design modifications that were hypothesized to address RU's issues. Moreover, in applications in which Trackman had to run inference in real-time or; had to deploy a model in short-notice, SU would prove useful.

## 6 Conclusion and Further Work

The objective of this work has been successfully achieved. The Residual-RELU model, presents a 15% improvement in test RMSE, as well as a reduction in parameter count, down to **1%** of the original baseline model. Moreover, the Squeeze-Excite Residual SILU model, achieves significant inference and training time reductions, at a still, **3%** better test RMSE and a relative parameter count of **2%**. A logical process for the emergence of these models has been also described; allowing for the reproduction of the models, as well as serving as a guideline for the architectural choices made.

However, we recognize that there is room for improvement and that there are some points that are left unresolved, setting the path for future work:

- Although the presented models beat the baseline models in the categories specified in the project goals, the consistency of the models wasn't desirable.
- Perhaps, a more exhaustive ablation study on the impact of all the tested hyperparameters, on the performance of the models, could have provided a more complete view of the a design choices made for the models. Time and resources availability to test the models were the main impediment in this regard.
- Further exploration of hyperparametrs and model designs could have help explain the instability of some models, as well as improve in terms of Trackman's goals definition.
- Further improvements reaching outside of the architecture itself can be explored; for example, a Knowledge Distillation framework(an idea implemented by another team in Trackmans's project), using our RU model as teacher, might give further reductions in model parameters while keeping similar performance.

**A notebook and set of scripts used for this project are available in a public github repository[3]. Note that the results are not fully reproducible due to confidentiality agreements with Trackman. The data provided has been synthetically created, with the same shape and format as the original one[4].**

## References

[1] J. Bowler. Eddy-current inversion for the determination of crack geometry. https://ieeexplore.ieee.org/document/757644. Accessed: 2024-12-13.

[2] Global average pooling explained. https://paperswithcode.com/method/global-average-pooling. Accessed: 2024-12-12.

[3] Artur Aah and Contributors. Trackman doppler, 2024. URL https://github.com/arturaah/Trackman_doppler. Accessed: 2024-12-21.

[4] OpenAI. Chatgpt: Assisting with ideation and code completion. https://chat.openai.com/, 2024. ChatGPT was used for ideation and code completion in this work.