Kölner Beiträge zur technischen Informatik
Cologne Contributions to Computer Engineering
Band 1/2025

# Ergebnisse der Workshops 2024 und 2025 des Forschungsschwerpunkts Vernetzte intelligente Infrastrukturen und mobile Systeme (VIMS)

Technology
Arts Sciences
TH Köln

**Impressum:**

Forschungsschwerpunkt Vernetzte intelligente Infrastrukturen und mobile Systeme
Technische Hochschule Köln
Fakultät für Informations-, Medien- und Elektrotechnik
Betzdorfer Str. 2
D-50679 Köln
tobias.krawutschke@th-koeln.de
Stand: Juni 2025
ISSN: 2193-570X

# Inhaltsverzeichnis

# Editorial

Mit diesem Band im Jahr 2025 wird die Veröffentlichung in der Reihe *Kölner Beiträge zur technischen Informatik* nach dem Band 1/2024 fortgesetzt. Die Fakultät für Informations-, Medien- und Elektrotechnik am Institute of Computer and Communication Technology (ICCT) ermöglicht Master Studierenden nicht nur aus dem Bereich der technischen Informatik eine Möglichkeit Ihre Forschung zu veröffentlichen, die im Rahmen von Forschungs- und Entwicklungsprojekten an der TH Köln und/oder bei Projektpartnern entstand.

Ziel ist es, die Ergebnisse laufender Arbeiten aus den Forschungs- und Entwicklungsaktivitäten des Forschungsschwerpunkts nach außen zu kommunizieren und Informatiker:innen und Informationstechniker:innen außerhalb des Forschungsschwerpunkts z.B. aus dem Kölner Raum einzuladen, neue Ergebnisse aus Wissenschaft und technischer Anwendung im Rahmen von *Cologne Open Science* zu publizieren.

In zwei Workshops wurden die hier veröffentlichten Themen diskutiert. Am 22.11.2024 wurden *Exploring transformer-based models for the basic task of text classification (Malte Autrata)* und *Adaptive Ramp Metering Control with Double Deep Q-Learning – A Simulation Study (Luca Schex, Leon Schex, Chunrong Yuan)* vorgestellt. Am 16.5.2025 wurde *DEEP-SEED: From Scratch to Ensemble – A Deep Learning Approach to Seedling Classification (Luca Uckermann)* vorgestellt.

Der Herausgeberkreis freut sich, diesen neuen Band der Reihe der Fachöffentlichkeit zur kritischen Prüfung und zur möglichen Mitwirkung vorlegen zu können.

Rainer Bartz
Andreas Behrend
Andreas Grebe
Tobias Krawutschke
Hans W. Nissen
Beate Rhein
René Wörzberger
Chunrong Yuan

# Exploring transformer-based models for the basic task of text classification

Malte Autrata, *malteautrata@googlemail.com*, *Accompanying lecturer: Prof. Dr. Beate Rhein*

***Abstract*—This paper evaluates the performance of transformer-based models (BERT, T5, and Llama 3) for classifying German newspaper articles into nine predefined categories. A dataset of 10,273 articles [1] was used to compare the models in terms of accuracy, training efficiency, and inference speed, highlighting the trade-offs between computational requirements and classification performance. BERT achieved a test accuracy of 90.27% with minimal training time, making it a practical choice for balanced performance and efficiency. Llama 3 delivered the highest accuracy at 92.86% but required significantly longer training times. The T5 model, while flexible in text-to-text tasks, slightly lagged in accuracy. These results underscore BERT's suitability for resource-constrained scenarios and Llama 3's potential in high-accuracy applications where computational resources are plentiful.**

***Index Terms*—LLM, large language model, transformer, text classification**

## I. Introduction

Text classification is a fundamental task in Natural Language Processing (NLP) that involves categorizing documents into predefined categories. This process is essential for managing and organizing large text corpora and supports applications such as fake news detection, email filtering, and e-commerce review analysis [2]. By automating text classification, businesses and organizations can streamline workflows and derive actionable insights [3].

Text classification tasks can be broadly categorized into off-the-shelf classification, zero-shot classification, and supervised learning using labeled data. This paper focuses on supervised learning, leveraging supervised machine learning models to compare performance and computational requirements. This approach offers key advantages, including independence from Application Programming Interfaces (APIs) and enhanced data privacy. The models analyzed include Bidirectional Encoder Representations from Transformers (BERT), Text-to-Text Transfer Transformer (T5) and Llama 3, all of which are based on the transformer architecture. This architecture has revolutionized NLP by achieving state-of-the-art results in tasks like machine translation while being more parallelizable than traditional approaches like recurrent neural networks [4]. Transformers' attention mechanism enables them to capture contextual relationships across a sequence by relating all positions within it.

The study evaluates these models using the 'Ten Thousand German News Articles Dataset', a collection of 10,273 articles categorized into nine classes: 'Web', 'Panorama', 'International', 'Wirtschaft', 'Sport', 'Inland', 'Etat', 'Wissenschaft' and 'Kultur' [1] [5]. The dataset presents challenges due to its imbalanced class distribution, with some categories such as 'Etat', 'Wissenschaft' and 'Kultur' being significantly underrepresented. By comparing model accuracy, training duration, and inference speed, this paper aims to identify the optimal model for different use cases, balancing performance and computational resource requirements.

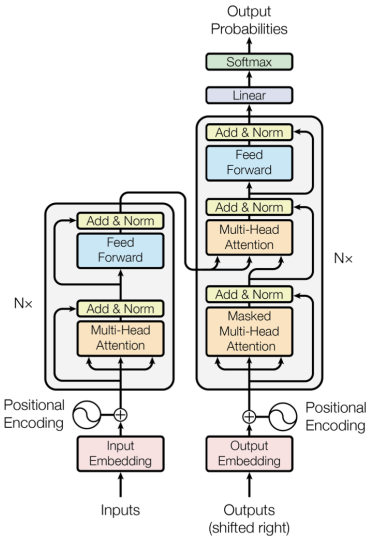



Fig. 1. The Transformer architecture consists of two main components: the encoder on the left and the decoder on the right. The encoder processes the input sequence and generates a set of representations, which are then passed to the decoder. The decoder uses these representations, combined with its own inputs, to establish attention mechanisms that connect the input and output sequences, enabling context-aware output generation [4].

## II. Models

This section provides an overview of the transformer-based models evaluated in this study: T5, BERT, and Llama 3. These models were selected because they represent distinct and state-of-the-art transformer architectures (shown in Fig. 1), are open-source, compatible with various frameworks, and collectively allow a comprehensive examination of performance across a spectrum of complexity (from foundational transformer models to a cutting-edge large language model).

### A. T5 base model

The T5 model employs an encoder-decoder architecture, transforming all tasks into a text-to-text format. Pre-trained

on diverse datasets using unsupervised learning, T5 can be fine-tuned for specific tasks by adapting the input-output format [6]. This flexibility in task framing makes T5 highly versatile, suitable for classification tasks. The base configuration includes 12 encoder and decoder layers, totaling 220 million parameters, with an embedding dimensionality of 768. During pre-training, the model processes sequences of up to 512 tokens [6]. T5 base achieved a General Language Understanding Evaluation (GLUE) score of 82.7, indicating robust classification performance [6].

### *B. BERT large model*

BERT utilizes an encoder-only architecture, capturing contextual information bidirectionally by considering both left and right contexts simultaneously. This capability makes BERT highly effective for tasks requiring deep language understanding, such as classification and question answering. Its encoder-only structure is well-suited for classification tasks where a representation of the entire sequence can be used to predict the class. However, each task requires adaption with the corresponding output layer [7]. The large variant of BERT consists of 24 encoder layers with 340 million parameters and an embedding dimensionality of 1,024. Like T5, it handles input sequences of up to 512 tokens during pre-training. BERT large scored 82.1 on the GLUE benchmark [7], slightly below T5 large, but it remains a robust choice for classification tasks.

### *C. Llama 3 8B model*

Llama 3 is a large language model (LLM) with a decoder-only architecture designed for auto-regressive tasks [8]. As the most complex model in this study, it is included to examine how well a LLM compares to more basic transformer architectures for text classification. Llama 3's larger context window (8,000 tokens) [9] and higher embedding dimensionality (4,096) [10] give it an advantage in capturing detailed contextual relationships. Pre-trained on multilingual datasets, with 5% of the data in non-English languages, Llama 3 is expected to perform well in multilingual and domain-specific tasks. Despite its size, quantization techniques and parameter-efficient fine-tuning methods, such as Low-Rank Adaptation (LoRA), enable its use on resource-constrained systems. The inclusion of Llama 3 allows for a direct comparison of accuracy and computational requirements between large-scale and more compact models.

### *D. Summary*

These models were chosen for their distinct characteristics, which make them collectively sufficient to address the text classification task:

1) **Open-Source Availability**: Ensures accessibility, adaptability, and transparency in implementation.
2) **Framework Compatibility**: Supports integration with popular machine learning libraries.
3) **Architectural Diversity**: Provides a spectrum of transformer designs:
   - **T5**: Encoder-decoder architecture for task versatility.
   - **BERT**: Focuses on deep bidirectional context understanding with an encoder-only design.
   - **Llama 3**: Adopts a decoder-only approach, emphasizing high accuracy in tasks requiring extensive context.
4) **Scalability**: The models vary in size and resource requirements, allowing for the evaluation of trade-offs between accuracy and efficiency.

This diversity allows a comprehensive comparison of their accuracy, training efficiency, and implementation complexity, as explored in subsequent sections.

## III. Implementation

This section details the implementation process for fine-tuning and evaluating the T5, BERT, and Llama 3 models on the text classification task. The models were trained and tested on the 'Ten Thousand German News Articles Dataset', with adjustments made to optimize performance and accommodate hardware constraints.

### *A. Training environment*

The models were trained on a server equipped with two NVIDIA Titan RTX GPUs, each with 24 GB of VRAM [11]. This hardware setup influenced choices such as model size, batch size, and optimization techniques. CrossEntropyLoss was used as the loss function, and the AdamW optimizer was employed for all training processes. Hyperparameters were tuned individually for each model to maximize performance.

### *B. Model-specific implementations*

*1) T5 base implementation:* To adapt the T5 model for text classification, the prefix 'Klassifiziere nachfolgenden Artikel in eine der folgenden Kategorien: Web, International, Etat, Wirtschaft, Panorama, Sport, Wissenschaft, Kultur oder Inland:' ('Classify the following article into one of the following categories: ...') was prepended to each input article, converting the task into a text-to-text format.
Key implementation details:

- **Tokenization**: Inputs were tokenized with a maximum sequence length of 512 tokens, truncating longer inputs.
- **Output Format**: The model generated class labels as textual outputs, restricted to five tokens for computational efficiency.
- **Training Configuration**: A learning rate of 5e-05, a batch size of 16, gradient clipping at 1.0, 2,000 warm-up steps, and 10,000 total steps were used.
- **Evaluation Criterion**: Outputs deviating from the predefined categories were considered incorrect.

*2) BERT large implementation:* The BERT model was fine-tuned using the deepset/gbert-large pre-trained variant, which is optimized for German-language tasks [12]. This version was selected because it is well-documented and widely adopted for German text. A classification head was added to the [CLS]

token representation to enable single-label classification.
Key implementation details:

- **Tokenization**: Inputs were tokenized with a maximum sequence length of 512 tokens, truncating longer inputs.
- **Model Architecture**: The classification head consisted of a feed-forward network on top of the [CLS] token representation with a hidden layer and 10% dropout to prevent over-fitting.
- **Training Configuration**: A learning rate of 2e-07, a batch size of 8, a maximum gradient normalization of 1.0 and 17,500 training steps were employed. No warm-up steps were included.

*3) Llama 3 8B implementation:* Given the large size of the Llama 3 model, modifications were made to fit it within the available hardware constraints. A classification head was added to the model output to enable it to perform single-label classification.
Key implementation details:

- **Tokenization**: Input sequences were capped at 1,500 tokens for practical training duration, despite the model's 8k-token context window capability.
- **Quantization**: Model weights were reduced to 8-bit integers during training to lower memory usage and computational requirements [13].
- **Model Architecture**: Low-Rank Adaptation of Large Language Models (LoRA) was applied to the attention and Multilayer Perceptron (MLP) modules, enabling efficient parameter updates with minimal additional overhead [14]. The classification head was added on top of the decoder output as fully trainable parameters.
- **Training Configuration**: A learning rate of 1e-04 was used for the first 3,000 steps and reduced to 1e-05 thereafter. Weight decay (0.01) was introduced after 1,200 steps to prevent over-fitting. A batch size of 4 was adopted, training was distributed across multiple sessions to allow hyperparameter adjustments, and 4,300 training steps were used. The LoRA rank was set to 8 with LoRA alpha set to 16 and a LoRA dropout to 0.01.

### C. Challenges and optimizations

- **Hardware Constraints**: The GPU memory limited the batch sizes and sequence lengths for larger models, necessitating techniques like quantization and LoRA for Llama 3.
- **Dataset Splitting**: A fixed test set and randomly split training and evaluation sets (20% evaluation split) were used for all models. While this approach maintains consistency in test-set evaluation, random splits could introduce minor variations in training dynamics.
- **Loss and Accuracy Tracking**: Loss and accuracy curves were monitored throughout training to identify convergence and stability issues, with adjustments made to hyperparameters as needed.

## IV. Results

This section presents and analyzes the performance of the T5, BERT, and Llama 3 models on the text classification task.

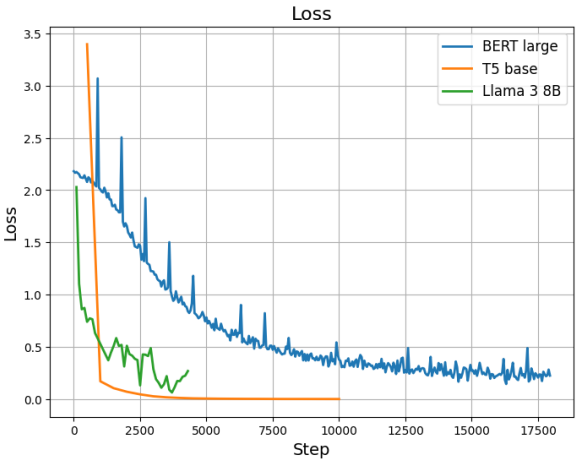



Fig. 2. Comparison of loss curves during training for the BERT large, T5 base, and Llama 3 8B models on the text classification task. The x-axis represents the number of training steps, while the y-axis denotes the loss.

The evaluation considers multiple metrics, including accuracy, precision, recall, F1 scores, training duration, and inference speed, to provide a comprehensive comparison. Figures and tables are used to illustrate key findings.

### A. Training dynamics

The training loss and accuracy curves (Fig. 2 and 3) for each model reveal significant differences in convergence behavior:

- **T5 base**: Converged rapidly, with loss stabilizing near zero within the initial training steps. Its token-based loss computation due to the text-to-text format may explain this behavior. Accuracy reached approximately 85% early and stabilized with minimal fluctuations.
- **BERT large**: Demonstrated slower and more oscillatory convergence, requiring over 10,000 steps to reach peak performance. The loss curve reveals spikes after every 1,000 steps, which could be the reason for reshuffling the dataset. These spikes become smaller as the training progresses and finally disappear completely after approximately 10,000 steps. Accuracy gradually increased, ultimately plateauing slightly below the final accuracy of Llama 3 at 90.64%.
- **Llama 3 8B**: Achieved the highest accuracy early in training, surpassing 90% within the first 2,000 steps. Despite minor fluctuations, its loss and accuracy stabilized efficiently, aided by hyper-parameter adjustments during training. The final evaluation accuracy reached a strong 92.70%.

### B. Evaluation metrics

Table I summarizes the evaluation metrics for each model.

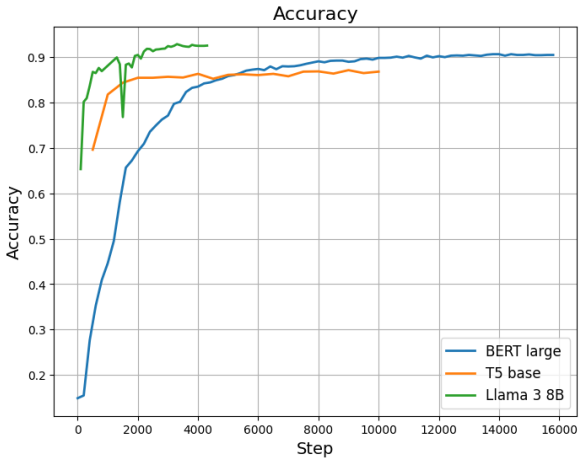



Fig. 3. Accuracy comparison during training for BERT large, T5 base, and Llama 3 8B models on a text classification task. The x-axis represents the number of training steps, while the y-axis indicates the accuracy.

TABLE I
TEST-SET RESULTS

| Metric | T5 base | BERT large | Llama 3 8B |
|---|---|---|---|
| test-set accuracy | 89.01% | 90.27% | 92.86% |
| test-set balanced accuracy | 89.08% | 90.46% | 92.41% |
| macro precision | 88.33% | 89.47% | 92.50% |
| micro precision | 89.01% | 90.27% | 91.95% |
| macro recall | 89.08% | 90.46% | 92.41% |
| micro recall | 89.01% | 90.27% | 92.50% |
| macro F1 score | 88.69% | 89.86% | 92.18% |
| micro F1 score | 89.01% | 90.27% | 92.41% |

**Key observations:**

- **Accuracy and precision**: Previous research on this dataset reported a accuracy of 90.5%, achieved using German-specific BERT variants [15]. In this study both BERT (90.27%) and Llama 3 (92.86%) demonstrated comparable or improved performance, indicating the strength of the implemented approaches. T5's accuracy of 89.01%, while slightly below the baseline, still reflects robust performance given it's flexibility and efficiency. Balanced accuracy metrics were closely aligned, reflecting consistent performance across both balanced and unbalanced datasets.
- **Generalization**: T5 exhibited inconsistent performance, with test-set accuracy (89.01%) higher than evaluation-set accuracy (87.13%), suggesting sensitivity to challenging samples in the evaluation set.

### C. Training and inference efficiency

Table II highlights the training and inference durations for each model.

TABLE II
EFFICIENCY

| Metric | T5 base | BERT large | Llama 3 8B |
|---|---|---|---|
| training time (minutes) | 140 | 448.16 | 3,183 |
| test-set inference time (minutes) | 0.43 | 0.72 | 3.6 |

**Insights:**

- **GPU utilization**: T5 and Llama 3 were trained and tested using two GPUs, allowing for faster processing, while BERT was trained on a single GPU due to its specific implementation setup. This discrepancy should be considered when comparing training efficiency across the models.
- **Flexibility in Llama 3 training**: While Llama 3 was trained with a classification head in this study, it is also possible to train it as a text-to-text model, similar to T5. This flexibility allows Llama 3 to adapt to a wider range of tasks depending on the specific requirements and resource availability.
- **Efficiency**: T5 required the least training and inference time, making it the most efficient model for rapid experimentation. Despite using only one GPU, BERT maintained competitive performance and training times, further demonstrating its practicality for resource-constrained environments. Llama 3's resource-intensive training reflects its complexity and parameter size.
- **Implementation complexity**: BERT's need for a classification head added some implementation overhead compared to T5. Llama 3 required advanced techniques like quantization and LoRA to mitigate its high resource demands.

### D. Confusion matrix analysis

The confusion matrices (Fig. 4) reveal model-specific challenges:

- All models struggled to distinguish between the 'International' and 'Panorama' categories, suggesting significant overlap in the dataset's features for these classes.
- Llama 3 consistently outperformed T5 and BERT across most categories, demonstrating its ability to leverage its larger context window and richer representations.

### E. Summary

While Llama 3 achieved the highest overall accuracy and precision, all models performed similarly in accuracy and balanced accuracy, with differences being minor. BERT provided a strong balance between performance and efficiency, emerging as the most practical choice for most use cases. T5's rapid convergence and simplicity make it ideal for tasks where quick implementation and lower resource usage are priorities. These results provide actionable insights into the trade-offs of transformer-based models for text classification.

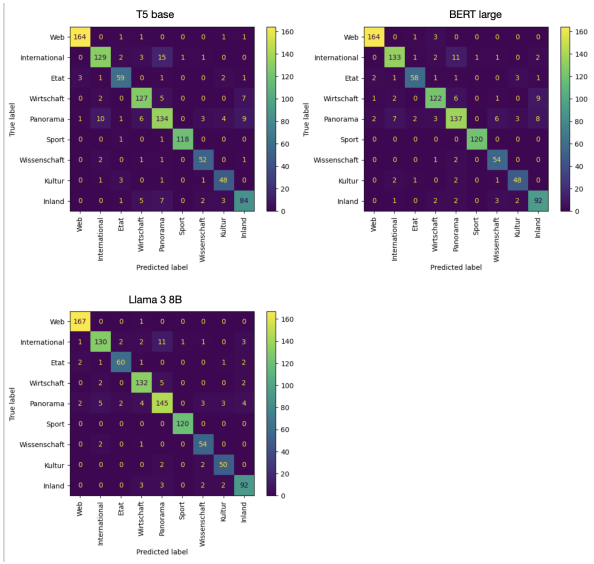



Fig. 4. Confusion matrices for the text classification models T5 Base, BERT Large, and Llama 3 8B, showcasing their performance across nine categories: 'Web', 'International', 'Etat', 'Wirtschaft', 'Panorama', 'Sport', 'Wissenschaft', 'Kultur', and 'Inland'. The rows represent the true labels, while the columns denote the predicted labels. The diagonal cells indicate correctly classified instances, with higher values reflecting better performance.

TABLE III
USE CASES

| Model | Use case |
|---|---|
| T5 base | When quick results combined with little effort are most important. |
| BERT large | Offers a strong balance between performance and efficiency. |
| Llama 3 8B | Delivers the highest accuracy, when computational resources and effort is no limiting factor. |

## V. CONCLUSION

This study compared the performance of three transformer-based models (T5, BERT, and Llama 3) on the task of classifying German newspaper articles into predefined categories. The results highlight the trade-offs between accuracy, computational efficiency, and resource requirements, offering valuable insights for selecting models in real-world applications, as summarized in table III.

BERT large emerged as the most balanced option, achieving a strong accuracy of 90.27% while maintaining relatively low training and inference durations. Its efficient encoder-only architecture and availability as a German-optimized variant make it particularly well-suited for scenarios with limited computational resources or strict time constraints. However, the need for a task-specific classification head adds complexity to its implementation.

The T5 base model demonstrated flexibility and ease of adaptation by re-framing the classification task as a text-to-text problem. While its accuracy (89.01%) was slightly lower than BERT's, T5 required less implementation effort and exhibited rapid training convergence, making it ideal for projects where

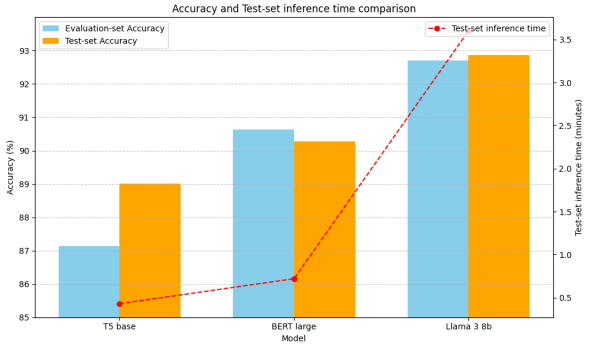



Fig. 5. Model comparison: All models deliver satisfactory results, with T5 base excelling in ease of training, BERT large offering the best trade-off of between inference time and accuracy, and Llama 3 8B achieving the highest overall accuracy.

simplicity and efficiency are prioritized over peak accuracy.

Llama 3 8B, the most resource-intensive model, delivered the highest accuracy at 92.86%. Its decoder-only architecture and large context window proved advantageous for capturing nuanced relationships within the data. However, the significantly longer training and inference durations, along with the need for advanced techniques like quantization and LoRA, limit its practicality to scenarios where computational resources are abundant, and achieving maximum accuracy is paramount.

The inclusion of Llama 3 allowed for a direct comparison between large language models and more compact transformer architectures. This comparison highlights that while larger models can offer superior performance, their benefits must be weighed against their resource demands, as highlighted in Fig. 5.

### A. Future work

Future research could explore:

1) Testing additional transformer architectures, such as A Robustly Optimized BERT Pretraining Approach (RoBERTa) or DistilBERT, to assess their trade-offs in performance and efficiency.
2) Addressing the dataset's class imbalance to improve performance across underrepresented categories.
3) Investigating techniques like data augmentation or active learning to further enhance model generalization.
4) Evaluating these models on more diverse and complex datasets to test their scalability and robustness.

In conclusion, the choice of a transformer-based model for text classification should be guided by the specific requirements of the application, balancing accuracy, resource availability, and ease of implementation. This study provides a framework for making such decisions, enabling practitioners to align model selection with their operational priorities.

## REFERENCES

[1] D. Schabus, M. Skowron, and M. Trapp, "One million posts: A data set of german online discussions," in *Proceedings of the 40th International*

*ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR)*, Tokyo, Japan, Aug. 2017, pp. 1241–1244.

[2] A. I. Kadhim, "Survey on supervised machine learning techniques for automatic text classification," *Artificial Intelligence Review*, vol. 52, no. 1, pp. 273–292, Jun. 2019.

[3] J. SINGH, "Natural language processing in the real world : Text processing, analytics, and classification." 2023.

[4] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds., vol. 30. Curran Associates, Inc., 2017. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf

[5] M. D. Timo Block, "Ten Thousand German News Articles Dataset," https://tblock.github.io/10kGNAD, mar 2019, accessed: 07-29-2024.

[6] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu, "Exploring the limits of transfer learning with a unified text-to-text transformer," 2023. [Online]. Available: https://arxiv.org/abs/1910.10683

[7] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," 2019. [Online]. Available: https://arxiv.org/abs/1810.04805

[8] Meta, "Introducing Meta Llama 3: The most capable openly available LLM to date," https://ai.meta.com/blog/meta-llama-3, Oct. 2024, accessed: 10-11-2024.

[9] Meta, "Llama 3 model card," https://github.com/meta-llama/llama3/blob/main/MODEL_CARD.md, 2024, accessed: 01-04-2025.

[10] NVIDIA CORPORATION & AFFILIATES, "Accelerating a hugging face llama 2 and llama 3 models with transformer engine," https://docs.nvidia.com/deeplearning/transformer-engine/user-guide/examples/te_llama/tutorial_accelerate_hf_llama_with_te.html, 2024, accessed: 09-23-2024.

[11] NVIDIA Corporation, "Titan rtx," https://www.nvidia.com/de-at/titan/titan-rtx/, accessed: 12-10-2024.

[12] B. Chan, S. Schweter, and T. Möller, "German's next language model," in *Proceedings of the 28th International Conference on Computational Linguistics*, D. Scott, N. Bel, and C. Zong, Eds. Barcelona, Spain (Online): International Committee on Computational Linguistics, Dec. 2020, pp. 6788–6796. [Online]. Available: https://aclanthology.org/2020.coling-main.598

[13] Meta, "Quantization | How-to guides," https://www.llama.com/docs/how-to-guides/quantization, Sep. 2024, accessed: 09-23-2024.

[14] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, and W. Chen, "Lora: Low-rank adaptation of large language models," *CoRR*, vol. abs/2106.09685, 2021. [Online]. Available: https://arxiv.org/abs/2106.09685

[15] deepset GmbH, "German BERT | State of the Art Language Model for German NLP," https://www.deepset.ai/german-bert, Jan. 2025, accessed: 01-04-2025.

# Adaptive Ramp Metering Control with Double Deep Q-Learning – A Simulation Study

Luca Schex
*Autonomous Systems Lab*
*TH Köln, Campus Deutz*
Betzdorferstr. 2
50679 Cologne, Germany
luca_jannis.schex@smail.th-koeln.de

Leon Schex
*Autonomous Systems Lab*
*TH Köln, Campus Deutz*
Betzdorferstr. 2
50679 Cologne, Germany
leon_nicolas.schex@smail.th-koeln.de

Chunrong Yuan
*Autonomous Systems Lab*
*TH Köln, Campus Deutz*
Betzdorferstr. 2
50679 Cologne, Germany
chunrong.yuan@th-koeln.de

***Abstract*— This work presents an innovative approach for ramp metering control on highways, wherein a double deep Q-Learning algorithm has been applied for finding optimal parameters to improve the traffic flow using Reinforcement Learning. Through automatic control of traffic lights at on-ramps, the entry of vehicles can be regulated intelligently so as to prevent congestion and make optimal use of available gaps, which is essential for the rush-hour traffic. The whole system was implemented in Python in a simulation environment, with models trained and tested using Pytorch. The system is able to control traffic lights at the on-ramp, by finding the optimal parameters for switching between green and red phases. We use a matrix to represent system state, which captures vehicle positions, speeds and the queue length at the on-ramp. In order to optimize the traffic flow, a reward function has been defined based on both the average vehicle speeds and the queue length. Compared to no control of traffic light, we have achieved a 47% reduction in the overall system travel time and a 27% reduction on the ramp in rush-hour scenarios. When the road network needs to deal with a substantially high traffic flow, the routing of vehicles on the highway and on the ramp can be regulated effectively using the proposed approach so that traffic congestion can be prevented effectively.**

***Keywords*— *Double Deep Q-Learning, Reinforcement Learning, Ramp Metering, Traffic Flow Optimization, Adaptive Traffic Control, Traffic Management, Urban Mobility***

## I. INTRODUCTION.

Nowadays, dealing with traffic congestion has become a critical issue worldwide, particularly in urban areas. Road systems has to be managed intelligently so as to reduce travel times, cut emissions and improve the overall traffic safety. In this work, we aim at optimizing highway traffic and propose an approach for the automatic control of ramp metering by adaptive learning of the optimal parameters based on a double deep Q-Learning algorithm.

A ramp metering system uses traffic lights installed at the on-ramp of a main roadway for the entry control of vehicles which are moving toward the on-ramp on a secondary road network. During periods with high traffic densities, traffic lights can be used to control the traffic flow on the highway so as to avoid situations where a large number of vehicles enter the main roadway at the same time. Since a large number of entering vehicles would require a relatively long sequence with more corresponding gaps for possible vehicle merging, it is necessary to break up vehicles formed at the traffic lights in the secondary road network into smaller groups or individual vehicles. Therefore, a ramp metering system has to be designed properly so that it can regulate the incoming traffic flow and try to prevent traffic congestion by making an optimal use of the gaps available on the highway.

One common method of traffic control uses a type of fixed-time operation, where traffic lights are switched according to a fixed schedule. The green phase is configured to allow either one or several vehicles to proceed simultaneously, with two vehicles at a time being a common setting to increase the outflow. The red phase is timed to minimize disruptions to the highway flow so as to keep the system stable. An enhancement of the fixed-time operation method is a simple traffic-dependent control based on demand and capacity. These systems operate without a specifically optimized control strategy, adjusting traffic light phases solely based on the current traffic volume [1].

Since the early 1990s, methods aiming at the improvement of the traffic-dependent control have been proposed. One example is the rule-based ALINEA algorithm [2]. By adjusting the inflow rate based on the traffic density measured downstream, just after the merge point, it tries to keep the flow rate on the main road close to its maximal capacity limit and prevents the highway from becoming overloaded and causing congestion. The measurement and calculation are performed at predetermined time intervals. The inflow rate will be increased if the measured vehicle density is low and it will be decreased if the measured density value becomes high.

Recently, thanks to the possibility of dynamic adjustment of control strategies, Reinforcement Learning (RL) has been widely used for ramp metering control [3][4]. In [3], RL has been used for controlling the number of vehicles entering the main highway from the ramp merging area. In [4], traffic cameras have been used so that information regarding the current traffic conditions can be extracted from video data for the control of ramp metering. Through a comparative experiment carried out in simulation, it has been shown that RL-based control has made a significant improvement over both the fix-time and rule-based control systems.

During rush hours, congestions can happen both on the main and a secondary road. In our viewpoint, it is very important to balance traffic flows on the main roadway and on the ramp so that we can avoid traffic jam on the main highway and prevent heavy traffic holdups into the secondary road network as well. This is particularly important in road networks with an above-average inflow from the on-ramp – such as interchanges at highway junctions or on-ramps near major intersections in urban metropolitan areas with a high traffic density. So instead of using RL for the single purpose of maximizing the traffic flow on the main highway, our goal is to develop a deep RL-based approach capable of rendering an optimal solution for all the vehicles involved.

In order to deal with this challenge problem, we have developed methods for finding the optimal input parameters for the RL algorithm. These parameters are essential for building the control mechanisms within RL as they help to determine how the dynamics of the rapidly changing actions of traffic lights should cope with the inertia of the reactive

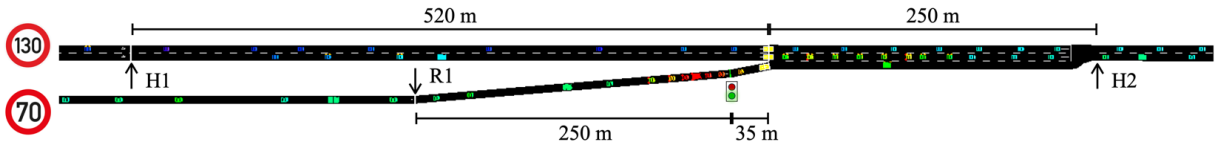



Fig. 1. The test scenario with a main highway and a ramp.

system with changes of traffic situations on both the highway and the ramp. In addition to fixing these optimal input variables, a further contribution of our work lies in the definition of reward function, which is indispensable for the RL algorithm to generate favorable system behaviors and making it converge toward the final goal of obtaining a dynamic balance among the flow rate on the highway, the queue length of the on-ramp and the flow rate on the on-ramp.

The rest of the paper is organized as follows: Section II deals with scenario definition and modeling for the ramp metering system. Section III discusses the motivation and concept of a system-level optimization. Details of the proposed learning algorithm including the definitions of the state, actions, the reward function as well as the network architecture will be presented in Section IV. Experimental results are evaluated and discussed in section V. And finally, Section VI summarizes the whole paper.

## II. Scenario Definition and Modeling

In order to realize and test the proposed control method, we use SUMO, an open-source traffic simulation tool [5]. For ramp metering, we have created a system with a two-lane highway and a ramp, as is illustrated in Figure 1. The sizes of the different sections of the highway and the ramp have been defined in such a way that the travel time on the highway segment from the point H1 to H2 would roughly equal the travel time from the point R1 of the on-ramp to H2 if the road were totally free.

As possible test scenario, we have also thought about using a highway with three lanes. However, the use of a two-lane one makes the underlying control system more challenging, as it could generate more severe traffic congestions. With the reduction of lanes, vehicles on the highway would have less chances for making rooms for the merging of incoming vehicles from the on-ramp.

The length of the acceleration lanes is set as 250 meters, a little longer than the usual one existed in real-world, due to the safety restriction of the SUMO simulator. In real-word practice, vehicles can often continue driving on the shoulder if merging is not immediately successful. However, this option does not exist in SUMO.

For traffic simulation, two types of vehicles are considered: cars and trucks. In order to generate realistic traffic behaviors, vehicles are allowed to exceed the speed limit. This has been achieved by using an individual speed factor, which is assigned to each vehicle with a standard deviation of 0.1 and a mean of 1.0, thereby creating a natural variance in traffic flow.

For the simulation of vehicles following a preceding vehicle, the Krauss Car-Following model is used for setting the speed of the following car so as to ensure safe and smooth traffic flow. The speed is determined based on a set of parameters including reaction time, braking force and safety gap.

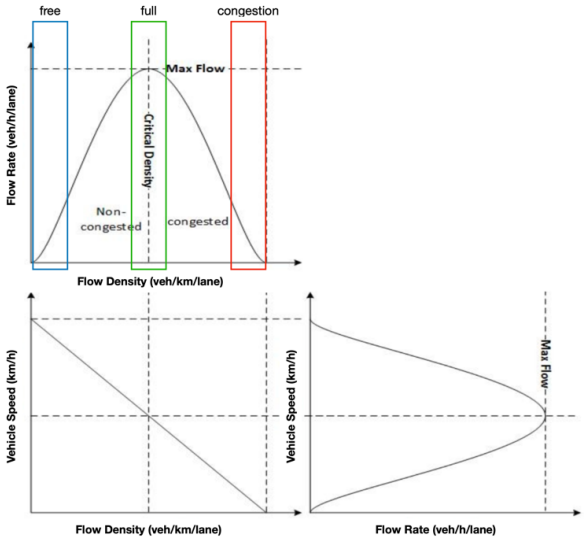



Fig. 2. Interacting factors of traffic flow.

In order to show the interaction and correlation of different parameters and their impact on the generation of traffic jam, we show in Figure 2 three diagrams whose axes are the rate of traffic flow represented as vehicles per hour and per lane (veh/h/lane), flow density measured as vehicles per kilometer and per lane (veh/km/lane), and vehicle speed calculated as kilometer per hour (km/h).

The curves shown in Figure 2 capture the interactions among these factors and give indications on the movement of vehicles and the tendency of potential change in road traffic. When flow density reaches the point of a traffic jam, as is shown by the red box, the flow rate drops to zero, which means that vehicles have come to a standstill. At the point of a critical density, as is shown by the green box, the flow rate reaches its maximum. This indicates that at the maximal flow rate, spacing between vehicles are optimal and this allows for the most efficient movement of vehicles on the highway. Below this critical threshold, both flow rate and flow density decrease, which indicates that the road capacity is underutilized. But above the critical density, flow rate decreases as traffic density increases, indicating that the movement of vehicles begins to interfere with each other in the traffic, leading to traffic congestion.

Based on the above discussion, it is clear that traffic congestion would occur when the flow density requested by vehicles moving on and toward a certain highway section would exceed its maximal density capacity. This is the reason, why it is called as the critical density of the highway.

## III. System-Level Optimization

Our control strategy lies in the optimization of a system-level travel time, reducing the travel times for all vehicles in the system, including those originally on the highway and

those on the ramp which want to enter the highway. Hence the total travel time of all vehicles driving through the system is used as the primary evaluation metric.

In order to calculate the system travel time, the travel time of each vehicle is determined as the difference between its arrival and departure times. For a vehicle on the highway route, this corresponds to the arrival time at point H2 minus the departure time at point H1, as is shown in Figure 1. Similarly, for a vehicle on the ramp route, it is calculated as the arrival time at point H2 minus the departure time at point R1. The system travel time is then obtained as the arithmetic mean of the travel times of all individual vehicles passing through the system.

For the purpose of visualization, an arithmetic mean of the travel times of all vehicles that have passed through the system during the last 100 simulation steps is calculated. This means, this value is updated every 100 simulation steps so that it can provide a visual representation of the dynamic traffic conditions.

In order to find optimal parameters for achieving a system-level optimization, we first define a constant scenario by setting the flow rate as 5000 veh/h for the main highway and as 2000 veh/h for the on-ramp. This generates a traffic flow where the flow rates on both the highway and the on-ramp are maintained close to the maximum flow, i.e., near the critical density. This is possible because vehicle positions and speeds in SUMO are determined by applying traffic rules based on speed limits and the minimum required distance between vehicles. With these two values set, it will lead to traffic situation with congestions, since the critical density of the highway section will be definitely exceeded after the merging.

If we start the traffic simulation with a permanently red traffic light at the on-ramp, no vehicles will come from the on-ramp, resulting in a traffic flow that is close to the maximum flow of the highway and yet below the critical density. The maximum flow on the highway section after the merging point could only be achieved if vehicles from the on-ramp could gradually merge onto the highway through an optimal inflow management. This is due to the fact that, both in reality and in SUMO, vehicles arrive or are spawned with slightly random gaps and speeds. These gaps allow vehicles from the on-ramp to merge seamlessly into the highway traffic flow. Hence after merging, the overall flow of the highway section is increased and yet without any disruption of the existing traffic flow on it.

Using SUMO, we have then made an analysis of the system behavior. We have compared the system-level travel times and flow rates obtained from three traffic scenarios, each with the constant traffic condition set above but with difference regarding the use of traffic lights and their phase changes. The three scenarios are:

- No control, equivalent to a traffic light permanently set as green
- Light switched with 5 seconds for green and 8 seconds for red
- Light switched with 50 seconds for green and 80 seconds for red

For each of the three cases, the simulation lasts 3600 seconds. Table 1 shows how the calculated travel times and flow rates change across all three situations. The table provides us with valuable information regarding traffic behaviors.

TABLE I. TRAVEL TIME, FLOW RATE WITH DIFFERENT TRAFFIC LIGHT CYCLES

| **Travel Time** | No-Control | 5/8 s Cycle | 50/80 s Cycle |
|---|---|---|---|
| System | 104.22 s | 68.49 s | 79.96 s |
| Highway | 107.18 s | 51.59 s | 66.11 s |
| Ramp | 97.30 s | 194.97 s | 152.93 s |
| **Flow Rate** | | | |
| System | 4070 veh/h | 4420 veh/h | 4264 veh/h |
| Highway | 2850 veh/h | 3899 veh/h | 3586 veh/h |
| Ramp | 1220 veh/h | 521 veh/h | 678 veh/h |

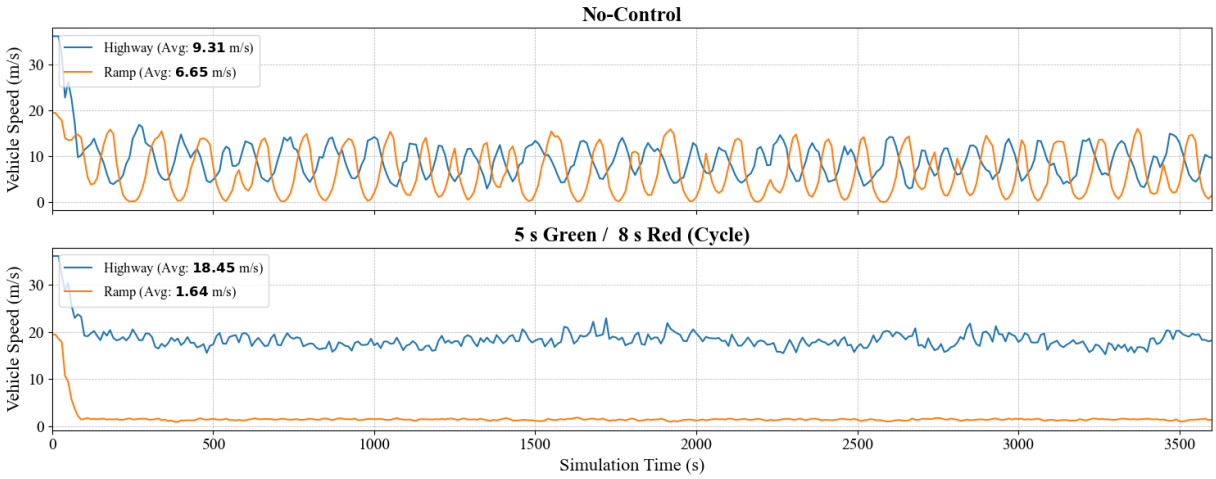
As presented in Table 1, traffic performance improves with more frequent traffic light switching: the average system-level travel time is 68.49 seconds with a switching cycle of 5 seconds green and 8 seconds red, and 79.96 seconds with a switching cycle of 50 seconds green and 80 seconds red. Both scenarios with switching traffic lights outperform the scenario with no control, where the average travel time is 104.22 seconds. The same conclusions can be derived from the flow rates. The flow rate with a switching cycle of 5 seconds green and 8 seconds red was reduced to 521 veh/h compared to 1220 veh/h with no control.

In a static scenario where the capacity of both road sections – the highway and the on-ramp – is maintained near the critical density, an improvement in the overall system travel time or an enhancement of the combined flow can only be achieved by prioritizing the road section that can achieve a higher throughput. In our case, this means regulating the on-ramp with a traffic light control to keep the highway flow as close to its maximum throughput as possible.

This leads to the conclusion that in order to keep the overall travel time of vehicles as low as possible, it is important to use traffic lights. And the time period used for switching the traffic lights should be kept small so that in each green phase, only a limited number of vehicles will enter into the main highway simultaneously.

If the traffic lights are not controlled properly, bulks will form, leading to wave-shaped backlogs on the merging line and the on-ramp, as can be observed in Figure 3. Here traffic speeds are compared between two cases: no control vs. traffic light with a switching cycle of 5 seconds green, 8 seconds red. The undulations of the speed curves in the upper part of Figure 3 is due to the fact that the vehicles shift frequently between the states of standstills and short-periods of start-ups. In the lower part of Figure 3, the speed curves are much more even and smooth.

Therefore, with a proper regulation of the traffic lights, vehicles from the on-ramp will enter the main highway in a sequential manner. If the traffic lights can be switched optimally between the red and green phase, fewer bulks will




form and as a consequence, no or fewer traffic jams would occur.

Since a static scenario with constant flow rate does not exist in reality and the flow rate on both the highway and the on-ramp varies, an optimization method has to be employed to control the traffic lights so that the flow would remain close to the critical density. It is therefore important to develop an intelligent control strategy which could let the system optimize toward the goal of maintaining a maximal flow on the highway while trying to reduce the length of the queue on the ramp as well. Of course, a trade-off between the flow rate on the highway and the queue length of the on-ramp, respectively the flow rate on the on-ramp, is desired.

## IV. Double Deep Q-Learning

For achieving efficient and stable ramp metering control, we use the double deep Q-Learning method. Unlike standard Deep Q-Learning (DQN), Double DQN utilizes its two Q-networks (online and target) by decoupling action selection (performed by the online network) from the value estimation of that action (performed by the target network). This mechanism specifically aims to reduce the overestimation of Q-values, leading to a stability improvement in the reinforcement learning process. In the following, details regarding this learning and optimization process will be presented.

### A. Actions

For a control system, a new decision is an action, which has to be made according to the change of the current system state. After the execution of an action, system state will change as the reaction of the action. In our problem of ramp metering control, we use a two-phase control of the traffic lights. As shown in Figure 4, the action set can be defined as

$$A = \{G, R\}, \tag{1}$$

where G stands for green and R stands for red. This means, within the RL-based control system, an action has to be chosen between two choices. Using a fixed time step, it determines the color of the traffic light for the next phase based on the current traffic state. In our case, a time step of 2 seconds has proven to be a good compromise: it is short enough to remain responsive, allowing only a single vehicle per traffic light phase onto the ramp, but long enough to detect changes in the traffic system and make the next decision accordingly.

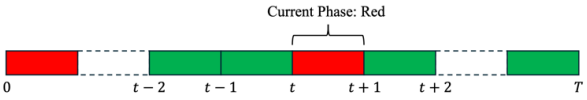



Fig. 4. Two-phase control.

### B. State

Since cameras can be used to monitor traffic nowadays, it can be used to obtain state information. Using SUMO, the current state of each vehicle including location, speed and distance travelled can be extracted directly.

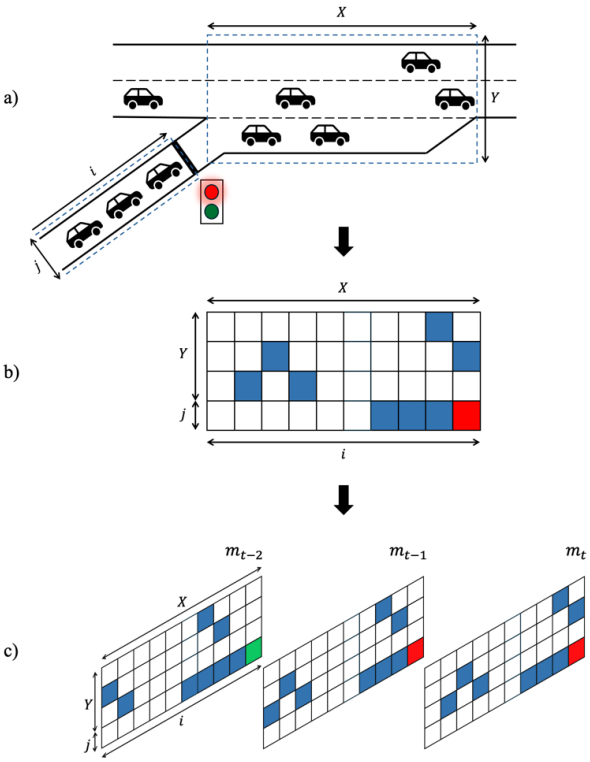



Fig. 5. Visualization of a) Control Area, b) Matrix $m_t$, c) Traffic State $s_t$

Figure 5 illustrates the process of state definition. Let the control area have a size of $X \cdot Y$, as shown in Figure 5 a), the positions of all vehicles $v(x_v, y_v)$ can be extracted from the simulation. The vehicle state at time step $t$ is represented in the form of a matrix $m_t \in \mathbb{R}^{X(Y+j)}$. As shown in Figure 5 b), $m_t$ also contains information on the signal state of the traffic light, which is positioned in the last column in row $j$. Integrated in row $j$ of the matrix $m_t$ is the length of queue formed at the traffic light. Since there are three vehicles waiting in the queue, there are also three occupied squares shown in color blue in the last row of the matrix $m_t$.

A single matrix is not sufficient to adequately capture the vehicle dynamics. Therefore, the matrices are stacked over 3 consecutive time steps, resulting in an overall system state $s_t$:

$$s_t = \{m_{t-2}, m_{t-1}, m_t\} \quad (2)$$

After having tested the state function based on the defined scenario in simulation, further adjustments have been made for performance optimization. It has been found that for the matrix $m_t$, it is better to take into accounts the length of each vehicle in the definition. This results in a state matrix which better quantify the dynamics of the vehicles. The final size of the $m_t$ matrix has been set as 4 x 251.

### C. Reward

Although the travel times of the vehicles are good parameters for evaluating the ramp metering system, they are not suitable to work as reward parameters, because they are strongly influenced by previous actions. Neither is the flow a suitable reward parameter. Since it is averaged over a certain period of time, it provides an inaccurate representation of the current state.

So instead, the speed on the on-ramp, speed on the section of highway above it, and the length of the queue before the on-ramp (i.e., number of vehicles waiting in front of the traffic light) are initially considered as the basis for defining the reward. Leading to the reward function $r_t$ determined as follows:

$$r_t = (\mu v_t + \omega q_t), \text{ where } \mu > 0,\ \omega < 0 \quad (3)$$

Here $v_t$ is the average speed on the on-ramp and the freeway section above, $q_t$ is the queue length before the on-ramp at time $t$. The weighting factors $\mu$ is positive for speed and the weighting factor $\omega$ is negative for queue length.

After experimented with dynamically changing traffic scenarios, it has been found out that the queue length alone is not sufficient to adequately represent the behavior on the section before the on-ramp. In situations where both the ramp and the highway are flooded with vehicles, the queue length on the ramp remains almost constant, which is equal to the maximal vehicle capacity determined by the length of the ramp. In such cases, the average speed of the vehicles on the ramp can be a more suitable and also reliable metric. Therefore, the average speed before the on-ramp was added as an additional factor to the reward function, leading to its final version defined by the following equation:

$$r_t = (\mu v_t + \omega q_t + \tau v r_t), \text{ where } \mu > 0,\ \omega < 0,\ \tau > 0 \quad (4)$$

Here $vr_t$ is the average speed before the on-ramp at time $t$ and $\tau$ is a positive weight factor.

### D. Architecture of the Neural Network

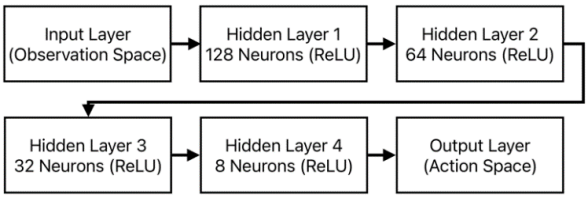



Fig. 6. Network architecture.

For the double deep Q-Learning algorithm, a Multilayer Perceptron (MLP) with fully connected layers was used for both the main network and the target network. As shown in Figure 6, the network begins with an input layer that corresponds to the observation space, which is equivalent to the system state matrix, with a size of 3×4×251=3012. The flattening operation is not performed explicitly in a dedicated flatten layer but is applied before the data are fed into the network. The data are then passed through several fully connected hidden layers with a decreasing number of neurons (128, 64, 32, 8) to extract essential features and process the complex patterns of the traffic environment. The final output layer corresponds to the action space with two possible actions: red or green, and provides the decision values for control. Each layer uses the ReLU activation function to enable non-linear learning.

## V. Experimental Evaluation

For the purpose of experimental study, we have simulated dynamic traffic situations with rush-hour conditions. Figure 7 shows the designed traffic scenario with changes in traffic flow during the morning rush hour between 7:50 and 8:50. The traffic flows on the highway and on the ramp, shown in color blue and red respectively, peak at 8:10 a.m., reflecting the expected traffic spikes during the morning rush hour. After 8:20, the flow rate gradually decreases, marking the end of the peak traffic period. However, traffic volume on the ramp remains relatively high.

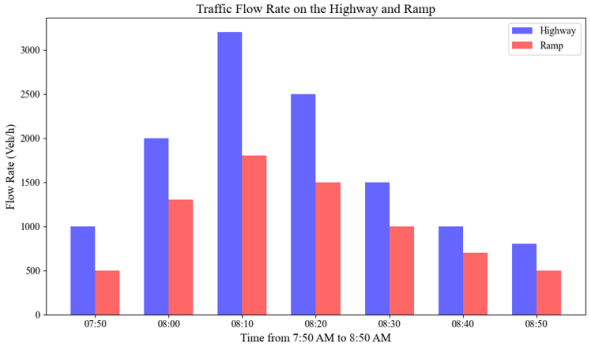



Fig. 7. Dynamic scenario with flow changes on both highway and ramp.

Using this basic traffic scenario with rush hour situations, we have compared our approach of deep Reinforcement Learning (deep RL) with two other control strategies: no control and fixed-time operation with a traffic light cycle of 5 seconds green and 5 seconds red.

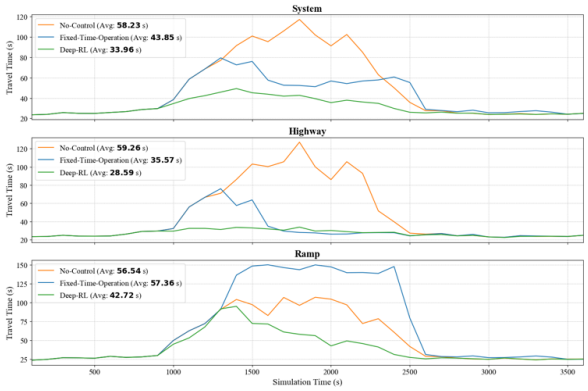



Fig. 8. Comparison of travel times.

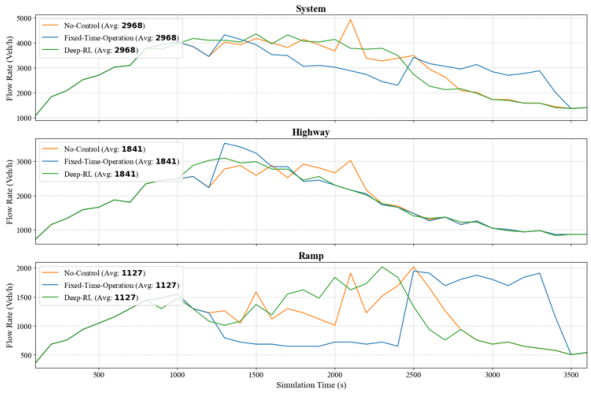



Fig. 9. Comparison of flow rates.

In the case of fixed-time operation, two vehicles will be allowed to pass during each green phase. A 5 second red phase also proves to be an effective choice, as it prevents excessive interference with the highway traffic flow, keeping the system stable (avoiding wave-shaped backlogs) while allowing a higher outflow from the on-ramp.

A comparison of the system-level travel times shown in Figure 8 highlights the significant improvements achieved through our deep RL approach. Compared to the no-control strategy, we were able to reduce the average system travel time by approximately 42%. Compared to the fixed-time operation, an improvement of around 25% was achieved. Our deep RL control also demonstrates clear advantages on the ramp: compared to the no-control strategy and the fixed-time operation, the average travel time for vehicles on the ramp has been reduced by about 24% and 26%, respectively.

One notable observation is that the average ramp travel times between the no-control and fixed-time operation scenarios differ by less than one second on average, even though the differences in the graph appear much larger. This is because each point on the curves shown in the diagram always represents the average value of the last 100 seconds, regardless of how many vehicles were included in the last measurement.

Looking at the ramp traffic flow in Figure 9 provides an explanation for this phenomenon. In the fixed-time operation, the traffic flow on the ramp is mostly delayed, meaning that most vehicles pass through the system only when highway traffic volume decreases. Since our system, as shown in Figure 1, has a length of 250 meters behind the traffic light, many vehicles wait beyond this boundary. Their waiting times are therefore not yet included in the measured ramp travel times, making the recorded values for the fixed-time operation appear lower than they actually are.

Expanding the considered system would thus make the advantages of our deep RL approach on the ramp even more apparent. Notably, our approach can shift the ramp traffic flow forward in time, allow more vehicles to pass earlier through the ramp and increase further the overall efficiency of the system. A similar effect can be also observed for the highway. Consequently, the efficiency of the entire system is enhanced.

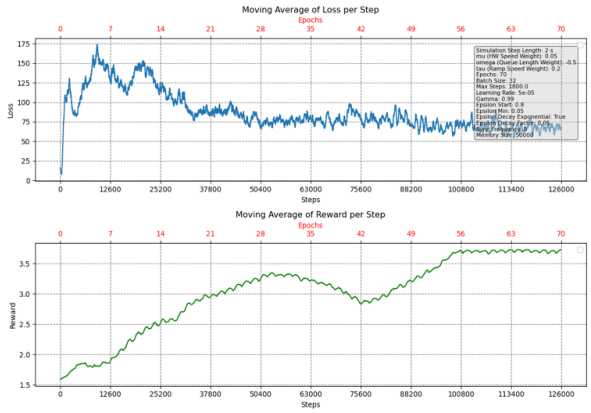



Fig. 10. The loss and reward functions.

By relieving congestion on the ramp earlier and enabling more efficient traffic control, the overall traffic flow is improved. This contributes to a noticeably more positive experience for road users passing through the system, as waiting times are reduced and overall travel times are shortened.

As shown in Figure 10, the decrease of the loss function can be clearly observed over the training period, indicating that the model is continuously learning from experience and improving its prediction accuracy. The moderately steep drop in loss at the beginning of training demonstrates that fundamental correlations in the data are quickly identified by the model. Subsequently, the loss curve flattens out, indicating that the learning process is stabilizing and approaching an optimum.

The reward function in the lower part of Figure 10 shows an overall upward trend, indicating the successful adaptation and learning of the model. The fluctuations and the temporary drop in reward may be attributed to the model's exploration of new strategies during the learning process, which initially prove to be suboptimal. However, overall, a convergence of the reward curve towards the end of training is observed, signaling increasing stability and efficiency in the behavioral strategies learned by the model.

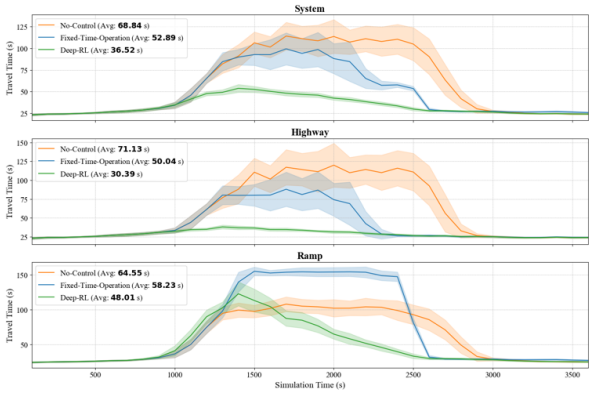



Fig. 11. Comparison of travel times with additional scenarios.

In order to evaluate the generalization capability of the trained model, it has been tested further with 25 new traffic scenarios. The traffic flow on the highway and the on-ramp, as is shown in Figure 7, was randomly varied by 100 to 500 vehicles per hour for each time period. Additionally, the random seed of the SUMO simulation was changed for each traffic scenario, resulting in different randomization of vehicle spawning within the simulation. The characteristic pattern of rush-hour has remained intact but with significant variations in intensity.

The test results shown in Figure 11 demonstrate that the model responds reliably to the newly generated traffic flows and even achieves better efficiency in terms of travel time savings for system users. On average, our deep RL approach has lowered the system-level travel time by approximately 47% compared to no control and by about 31% compared to fixed-time operation. Regarding the travel time on the ramp, we observed an average improvement of approximately 27% compared to no control and about 18% compared to fixed-time operation. As previously explained, extending the underlying system would further highlight the advantages of our deep RL approach on the ramp.

## VI. Conclusion

In summary, we have proposed an approach for the control of a traffic light based the double deep Q-Learning algorithm. Using the realized system, it is possible to mitigate the negative effects of vehicle merging behavior, which occurs frequently on highway on-ramps. The results show that in dynamic situations representing rush-hour scenarios and compared to no control, the system travel time and the travel time on the ramp can be reduced by an average of 47 % and 27 % respectively.

Capable of regulating the traffic flow on the ramp at the point where traffic density threatens to exceed critical values, this method can be used to effectively prevent the formation of wave-shaped traffic jams on the highway. Through intelligent traffic light management, seamless merging of vehicles from the on-ramp into the flowing traffic on the highway is enabled. Thanks to the intelligent control of traffic lights, abrupt braking and acceleration can be minimized, as is clearly reflected by the smoothness of speed curves. This leads to the cut of travel times for all road users and the increase of the overall efficiency of the whole traffic network.

This research lays the foundation for future work, where the proposed approach can be tested and refined in more realistic environments with multiple on- and off-ramps. Its integration into real-world traffic management systems would make a significant contribution by helping to reduce traffic congestion, improve traffic efficiency, enhance road safety and diminish emissions.

## References

[1] European ITS Platform (EU EIP), “Reference Handbook for Harmonised ITS Core Service Deployment in Europe,” October 2021. Available: https://www.its-platform.eu/achievement/reference-handbook. [Accessed: May 03, 2024].

[2] M. Papageorgiou, H. Hadj-Salem, J. M. Blosseville, “ALINEA: A local feed back control law for on-ramp metering,” *Transportation Research Record*, No. 1320, 1991, pp. 58-64.

[3] A. Fares and W. Gomaa, “Freeway ramp-metering control based on reinforcement learning,” *11th IEEE International Conference on Control & Automation (ICCA)*, Taichung, Republic of China, June 2014, pp. 1226-1231.

[4] B. Liu, Y. Tang, Y. Ji, Y. Shen, and Y. Du, “A deep reinforcement learning approach for ramp metering based on traffic video data,” *Journal of Advanced Transportation*, Oct 2021. Available: https://onlinelibrary.wiley.com/doi/10.1155/2021/6669028. [Accessed: May 10, 2024].

[5] German Aerospace Center (DLR), “SUMO user documentation,” Available: https://sumo.dlr.de/docs/index.html. [Accessed: May 08, 2024].

# DEEP-SEED:
# From Scratch to Ensemble – A Deep Learning Approach to Seedling Classification

Luca Uckermann
*Faculty of Information, Media and Electrical Engineering*
*Institute of Media and Imaging Technology*
*TH Köln – University of Applied Sciences*
*Cologne, Germany*
*luca.uckermann@th-koeln.de*

***Abstract*—This paper evaluates and compares three deep learning (DL) approaches for plant seedlings classification using a dataset consisting of *4750* images of *12* different plant species. Several DL approaches were considered, including a custom convolutional neural network (CNN) trained from scratch, a pre-trained CNN "ResNet-18" and a pre-trained vision transformer (ViT) "vit-base-patch16-224" fine-tuned for the task at hand. To address the challenges of data scarcity and class imbalance, extensive data augmentation techniques such as random rotations, flips and color jittering were employed. Results showed that transfer learning with ResNet-18 outperforms the custom model, achieving a mean F1-score (micro-averaged) of 0.961 on the test set. The custom CNN, still achieved a competitive F1-score of 0.927, demonstrating that even smaller locally trained architectures can be viable if carefully designed and thoroughly regularized. While the ViT model achieved the highest F1-score of 0.967, an ensemble combining the predictions of all three models outperformed the single models with a score of 0.971. Finally, potential improvements are outlined, including deeper architectures, synthetic image generation and interpretability measures, to further improve seedling classification performance.**

***Index Terms*—Machine learning, Image classification, Convolutional neural networks, Vision transformers, Transfer learning**

## 1. Introduction

This section provides an overview of the challenge, including its context, problem definition, dataset overview and key challenges.

### 1.1. Context and Background

The "Plant Seedlings Classification" challenge, hosted on Kaggle [1], presents a real-world problem central to modern agriculture: accurately identifying the species of young seedlings from digital images [2] and distinguishing between weeds and crops [3].

The dataset described in [4] contains images of approximately *960* unique plants, representing *12* different species (see Fig. 1). Each image captures a seedling at different growth stages and under different conditions, reflecting the complexities found in real-world agricultural environments. These conditions include differences in lighting, background soil patterns and subtle phenotypic variations that can blur the lines between certain species.


Figure 1. One sample image for each class in the dataset

Fig. 1 shows one sample image for each class in the dataset, illustrating the different species. The images vary in background, lighting and growth stage, highlighting the challenges of visual similarity across species.

The evaluation metric of the competition is a mean (micro-averaged) F1-score, which encourages balanced performance across all classes $C$ [5]–[8]:

$$\text{Precision}_\mu = \frac{\sum_{i=1}^{C} TP_i}{\sum_{i=1}^{C} (TP_i + FP_i)} \tag{1}$$

$$\text{Recall}_\mu = \frac{\sum_{i=1}^{C} TP_i}{\sum_{i=1}^{C} (TP_i + FN_i)} \tag{2}$$

$$\text{F1}_\mu = \frac{2 \times \text{Precision}_\mu \times \text{Recall}_\mu}{\text{Precision}_\mu + \text{Recall}_\mu} \tag{3}$$

where $TP_i$, $FP_i$ and $FN_i$ are the true positive, false positive and false negative counts for class $i$, respectively and $C$ is the set of all classes. The mean F1-score (3) is a balanced measure that considers both precision (1) and recall (2) across all classes, making it a suitable evaluation metric for multi-class classification tasks.

### 1.2. Problem Definition and Objectives

The core objective of the challenge is to build an automated classification model that can take a seedling image as input and accurately predict its species. The following points summarize the task:

1) **Input:** Train set of *4750* images of plant seedlings.
2) **Output:** Classification label for each image, indicating the species of each plant seedling.
3) **Goal:** High classification performance as measured by equation 3.

The challenge is to develop a model that can generalize well across different species, even when faced with variations in lighting, background and growth stages. A significant risk when dealing with deep learning (DL) models is overfitting, especially when the dataset is relatively small [9, Chapter 1] [10, Chapter 5]. Therefore, the model must be designed to learn robust features that can generalize well to unseen data (see 1.4).

### 1.3. Dataset Overview

For a better understanding of the dataset, a brief overview of the class distribution and sample images is provided below:

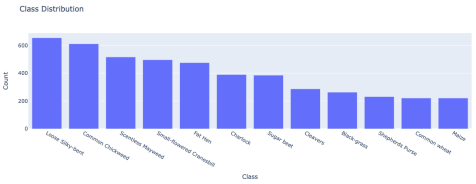



Figure 2. Class distribution of the *4750* training images

Fig. 2 shows the distribution of classes in the train dataset, with each bar representing the number of images per class. The dataset is imbalanced, with some classes having significantly fewer samples than others. This imbalance can pose a challenge for model training, as the model may struggle to learn the features of underrepresented classes effectively. The most common classes are "Loose Silky-bent" (*654*) and "Common Chickweed" (*611*), while the least common classes are "Common wheat" and "Maize" (both *221*).

### 1.4. Key Challenges

Developing robust classification models for this task is not trivial. There are several challenges:

1) **Inter-Class Similarity:** Certain seedlings can look strikingly similar, making it difficult for both humans and machines to distinguish between them.
2) **Intra-Class Variability:** Even within a single species, seedlings can vary significantly in appearance due to differences in growth stage, lighting and background. This variability challenges models to learn consistent features that generalize well.
3) **Data Limitations:** With approximately *960* unique plants, the dataset could be considered modest for training DL models from scratch. While data augmentation can help to some extent (see 3.3), the relatively small dataset may still limit the capacity of models that can be effectively trained without overfitting.
4) **Model Architecture Capacity:** Choosing the right model architecture, whether a custom convolutional neural network (CNN) [11] trained from scratch or a pre-trained deep CNN / Vision Transformer (ViT) [12], to learn complex visual features. Deeper models can capture more nuanced differences, but they can also be harder to train and require careful regularization to prevent overfitting (see 3.3).

By clearly understanding these challenges and the broader context, model architectures can be proposed that address these difficulties. The following chapters discuss the strategies for model design, training optimization, model evaluation and analysis of results, ultimately leading to the approach that best addresses the core challenge of differentiating between plant seedling species. Finally, the conclusion summarizes the key findings and suggests potential areas for future research.

## 2. Model Architecture Design

Python and Jupyter notebooks were used to implement the models and train them on the plant seedlings dataset. The code was organized into separate notebooks for each model, allowing for easy experimentation and comparison of different architectures. Libraries such as PyTorch, Scikit-learn, NumPy and Pandas were used for data manipulation, model training and evaluation.

To achieve deterministic results and reproducibility, the random seed *42* was set at the beginning of each notebook. This ensured that the same random initialization was used for each run, leading to consistent results across different experiments:

```
RANDOM_SEED = 42

seed(RANDOM_SEED)
np.random.seed(RANDOM_SEED)

torch.manual_seed(RANDOM_SEED)
torch.cuda.manual_seed_all(RANDOM_SEED)

torch.backends.cudnn.deterministic = True
torch.backends.cudnn.benchmark = False
```

Listing 1. Settings to achieve deterministic results and ensure reproducibility.

The random seed was set for the Python, NumPy and PyTorch random number generator. Additionally, the cuDNN backend was set to deterministic mode to ensure that the results are reproducible on the GPU.

## 2.1. Guessing Baseline

As a starting point and to get familiar with the dataset and the Kaggle competition, a simple guessing baseline was implemented. The baseline assigns the most frequent class label to all test samples. This approach provided a lower bound on model performance and served as a reference point for evaluating the effectiveness of more sophisticated models. The head of the submission file is shown below:

```
file,species
1b490196c.png,Loose Silky-bent
85431c075.png,Loose Silky-bent
506347cfe.png,Loose Silky-bent
7f46a71db.png,Loose Silky-bent
668c1007c.png,Loose Silky-bent
...
```

Listing 2. Head of guessing baseline submission file.

In this case, all *794* test samples were assigned the class label “Loose Silky-bent”, which is the most frequent class in the training dataset. The F1-score of this baseline is **0.14105**.

## 2.2. Custom CNN

The custom CNN architecture was designed (from scratch) to capture features relevant to seedling classification, while being lightweight enough to be effectively trained locally on the given dataset. The model consists of a series of convolutional and pooling layers followed by fully connected layers to learn features hierarchically and make the final class prediction.

As shown in Fig. 3, the network begins with a series of convolutional layers (blue), with the number of filters gradually increasing from *16* to *256*. These convolutional layers, each followed by a Rectified Linear Unit (ReLU) activation, extract spatial features such as edges, textures and patterns from the images. To reduce spatial dimensions and computational complexity, max-pooling layers (gray) are applied after each convolutional block to focus on the most salient features.

After the convolutional and pooling stages, the feature maps are flattened into a 1D vector that serves as the input to the fully connected layers (blue). The first fully connected layer has *128* neurons and captures high-level abstract features, while the final fully connected layer maps these features to the *12* target classes (green), producing the class probabilities.

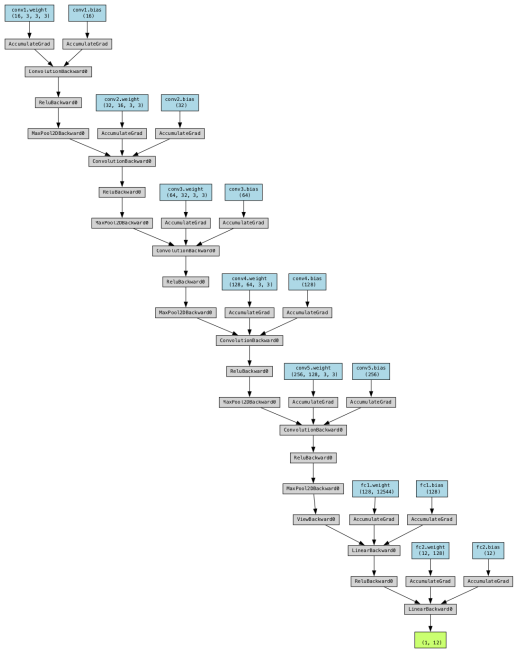


Figure 3. Custom CNN architecture, consisting of convolutional and pooling layers followed by fully connected layers.

The final custom CNN model has approximately *2 million* parameters, making it lightweight and computationally efficient compared to the following architectures. The model architecture is designed to capture relevant features for seedling classification while being suitable for training on a moderate-sized dataset.

## 2.3. Pre-trained CNN

As an alternative to training and building a custom CNN from scratch, a pre-trained CNN can be used to leverage learned features from a large dataset. The pre-trained model ResNet-18 [13] was used as a feature extractor, where the final classification layer was replaced with a new fully connected layer to predict the *12* plant seedling classes:

```python
from torchvision import models
from torch.nn import Linear

model = models.resnet18(
    weights=models.ResNet18_Weights.DEFAULT
)
model.fc = Linear(
    in_features=model.fc.in_features,
    out_features=len(dataset.classes),
)
```

Listing 3. Replacing the final classification layer of a pre-trained ResNet-18 model.

The `ResNet18` model has been pre-trained on the ImageNet dataset [14] and has shown strong performance on a variety of computer vision tasks [13]. By using a pre-trained model, the network can leverage the learned features from ImageNet to improve performance on the plant seedlings dataset. The final classification layer was replaced to adapt the model to the specific classification task.

This pre-trained CNN model has approximately *11 million* parameters, making it deeper than the custom CNN. However, fine-tuning the weights allow the model to learn more complex features and hopefully achieve better performance on the plant seedlings dataset.

## 2.4. Pre-trained ViT

Another approach is to use a ViT as the backbone architecture. The ViT model has been pre-trained on ImageNet-21k [15] (including plants/crops) and then fine-tuned [16] on the plant seedlings dataset. The final classification head was replaced with a new linear layer to predict the *12* plant seedling classes:

```python
import timm
import torch

model = timm.create_model(
    "vit_base_patch16_224",
    pretrained=True,
    num_classes=num_classes
)
model.head = torch.nn.Linear(
    model.head.in_features,
    num_classes
)
```

Listing 4. Replacing the final classification layer of a pre-trained ViT model.

Instead of fine-tuning the entire model, the pre-trained weights of the `vit_base_patch16_224` model [17] were frozen and only the classification head was trained on the plant seedlings dataset. Transfer learning leverages the powerful feature extraction capabilities of the pre-trained model while adapting the final layer to the specific classification task [9, Chapter 6]. Furthermore the computational cost is reduced compared to training the entire model from scratch or fine-tuning all layers:

```python
for param in model.parameters():
    param.requires_grad = False

for param in model.head.parameters():
    param.requires_grad = True
```

Listing 5. Freezing the pre-trained ViT backbone and training only the classification head.

The ViT model has approximately *86 million* parameters, but only *9,228* of these are adapted in the experiments. This makes the model computationally efficient, while still benefiting from the powerful feature extraction capabilities of the pre-trained ViT model.

## 2.5. Ensemble

Several challenges have shown that an ensemble of models can often outperform individual models, especially when inference time is not a primary concern [13], [18]. A final ensemble was created by combining the predictions of all three models (custom CNN, pre-trained CNN, pre-trained ViT) using a weighted average. The weights were determined based on the performance of each model on the test set:

```python
import torch.nn.functional as F

model_custom_cnn.eval()
model_resnet.eval()
model_vit.eval()

w_custom_cnn = 0.25
w_resnet = 0.25
w_vit = 1 - w_custom_cnn - w_resnet

...

probs_custom_cnn = (
    F.softmax(model_custom_cnn(images), dim=1)
)
probs_resnet = (
    F.softmax(model_resnet(images), dim=1)
)
probs_vit = (
    F.softmax(model_vit(images), dim=1)
)

probs_ensemble = (
    w_custom_cnn * probs_custom_cnn +
        w_resnet * probs_resnet +
            w_vit * probs_vit
)

_, preds = torch.max(probs_ensemble, 1)
```

Listing 6. Ensemble predictions using a weighted average.

The ensemble combines the strengths of each individual model to improve overall performance and robustness. By averaging the predictions of multiple models, the ensemble can reduce the impact of individual model weaknesses and provide more reliable predictions (*wisdom of the crowd* [19, Chapter 7]). As the ViT model achieved the highest performance on the test set, it is assigned the highest weight in the ensemble (*0.5*), while the custom CNN and ResNet models are assigned equal weights (both *0.25*).

### 2.6. Summary

TABLE 1. MODEL PARAMETERS AND SIZE OF THE CUSTOM CNN, PRE-TRAINED CNN AND PRE-TRAINED VIT.

| Model | Total Params | Trainable Params | Total Size (MB) |
|---|---|---|---|
| Custom CNN | 1,999,916 | 1,999,916 | 34.91 |
| Pre-trained CNN | 11,182,668 | 11,182,668 | 106.02 |
| Pre-trained ViT | 85,655,820 | 9,228 | 806.35 |

Table 1 shows the total number of parameters and the size of each model. The custom CNN has the smallest number of parameters and size, making it lightweight and computationally efficient. The pre-trained CNN has a larger number of parameters, while the pre-trained ViT has the largest, but only a small fraction of them are trainable during the experiments. This allows for efficient training while still benefiting from the powerful feature extraction capabilities of the pre-trained model.

## 3. Training Optimization Strategies

This section describes the training algorithms, learning rate schedules and regularization techniques used to improve model generalization and prevent overfitting.

### 3.1. Training Algorithms and Optimizers

All models were trained using the Adam optimizer [20] with a learning rate of *0.001*. The Adam optimizer is a popular choice for training deep neural networks due to its adaptive learning rate mechanism and momentum-based updates. A weight decay of *1e-4* was applied to regularize the model and prevent overfitting (see 3.3).

### 3.2. Learning Rate Schedules

To adjust the learning rate during training, a learning rate scheduler was used to reduce the learning rate by a factor of *0.5* if the validation loss did not improve for *2* epochs. This technique helps the model converge more effectively by gradually reducing the learning rate as it approaches a local minimum.

### 3.3. Regularization Techniques

To prevent overfitting and improve generalization, several regularization techniques were applied during training:

- **Weight Decay:** L2 regularization with a weight decay of *1e-4* was applied to the optimizer to penalize large weights (see 3.1).
- **Dropout:** A dropout layer with a dropout probability of *0.5* was added after the fully connected layer to regularize the model and prevent co-adaptation of neurons.
- **Data Augmentation:** Various data augmentation techniques such as random rotations, flips and color jittering were applied to the training images to increase the diversity of the training set and improve the robustness of the model (see Fig. 4).

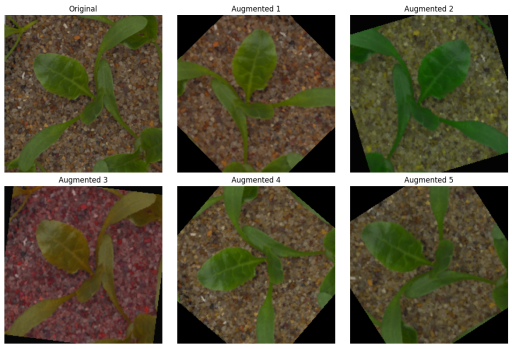



Figure 4. Original image (top left) and five augmented versions.

Fig. 4 shows one original image of the given dataset and five augmented versions. The augmentations include random rotations, flips and color jittering, which help the models learn more robust features and improve generalization to unseen data. During training those augmentations were applied randomly to each image, while the original images were used for validation. This approach allowed the model to learn from a more diverse set of training samples without introducing bias in the validation process.

## 4. Model Evaluation and Validation

This section describes the evaluation framework used to assess the performance of the models during training. It includes a discussion of the validation framework, performance metrics and results obtained from the models.

### 4.1. Validation Framework

To evaluate the performance of the models during training, the dataset was split into training and validation sets using a stratified split with a ratio of *80:20*. The validation set was used to monitor performance during training and prevent overfitting. Since the dataset is imbalanced, a stratified split was used to ensure that the class distribution in the training and validation sets is similar. This prevents the model from overfitting the training set and ensures that it generalizes well to unseen data. This split results in a training set of *3800* and a validation set of *950* samples.

Since the Kaggle challenge does not provide the labels for the test set, the validation set served as a proxy to evaluate the performance of the model on unseen data (not used during training). Validation loss and accuracy were monitored during training to assess the convergence and generalization capabilities of the models. Due to computational constraints, only one run per model was performed:

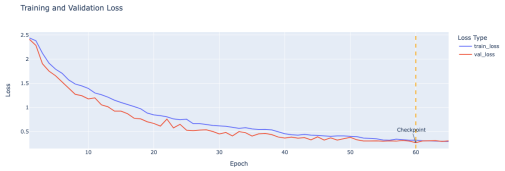



Figure 5. Training and validation loss (custom CNN)

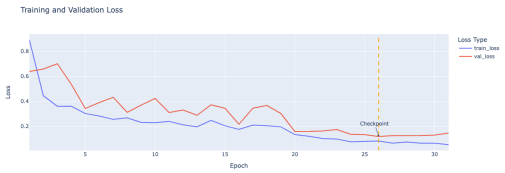



Figure 6. Training and validation loss (pre-trained CNN)

Fig. 5, Fig. 6 and Fig. 7 show the training and validation loss curves for the custom CNN, pre-trained CNN and pre-trained ViT models, respectively. Each curve illustrates the learning dynamics of the model over successive epochs.

The custom CNN demonstrated a gradual reduction in both training and validation loss, converging steadily around epoch *60*. This indicates effective learning without overfitting, as the training and validation loss curves remained closely aligned. One could argue that the model could benefit from increased capacity and further training to improve performance.

The pre-trained CNN showed faster convergence, with training and validation loss stabilizing around epoch *26*. This faster convergence reflects the advantages of fine-tuning, as the model uses pre-trained weights for feature extraction. Similarly, the pre-trained ViT, which also converged rapidly within *25* epochs, demonstrates the effectiveness of using pre-trained models and transfer learning for this classification task. However the gap between training and validation loss suggests that the model could benefit from additional regularization to improve generalization. In particular, the pre-trained CNN began to clearly overfit the data after epoch *30*.

The inclusion of a checkpoint in each figure highlights the point at which the model achieved the lowest validation loss, indicating optimal performance and serving as a reference for saving the best model state. The loaded checkpoints are at epoch *60*, *26* and *25* for the custom CNN, pre-trained CNN and pre-trained ViT, respectively.

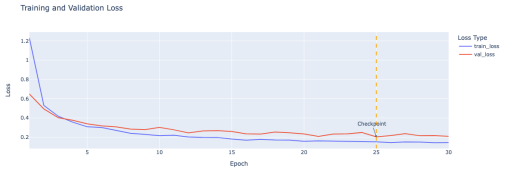



Figure 7. Training and validation loss (pre-trained ViT)

## 4.2. Performance Metrics

In addition to losses, validation accuracy was tracked during training to monitor performance of the models. Accuracy is calculated as the ratio of correctly predicted samples to the total number of samples in the validation set:

$$\text{Accuracy} = \frac{\text{Number of Correct Predictions}}{\text{Total Number of Samples}} \tag{4}$$

The accuracy (4) provides a simple and intuitive measure of performance on the validation set:

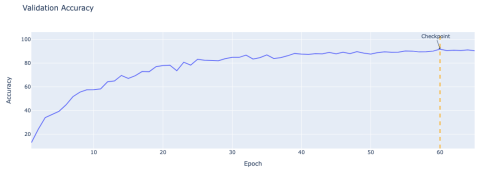



Figure 8. Validation accuracy (custom CNN)

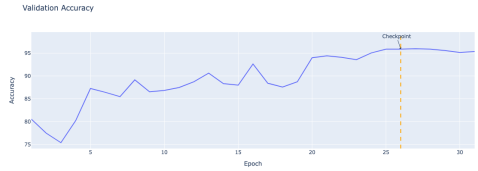



Figure 9. Validation accuracy (pre-trained CNN)

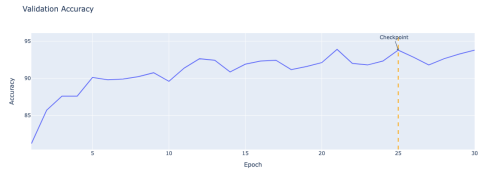



Figure 10. Validation accuracy (pre-trained ViT)

Similar to the loss curves, Fig. 8, Fig. 9 and Fig. 10 show the validation accuracy of the custom CNN, pre-trained CNN and pre-trained ViT models, respectively. The accuracy curves provide insight into the ability of the model to correctly classify the validation samples over successive epochs. The figures highlight the information from the loss curves, showing that the pre-trained models converged faster and achieved higher accuracy compared to the custom CNN.

But again, the custom CNN showed a steady and smooth increase in accuracy over time, indicating that the model continues to learn and improve its performance. The selected checkpoints are the same as for the loss curves, indicating the load state of the model with the lowest validation loss, which also corresponds to high accuracy.

## 5. Results and Analysis

This section presents the results of the models on the validation and test sets. The performance of each model is evaluated using various metrics, including training and validation loss, validation accuracy and test F1-score. The results are compared to assess the effectiveness of different architectures and training strategies. Additionally, qualitative results are provided to illustrate predictions of the models and highlight areas for improvement.

### 5.1. Quantitative Results

TABLE 2. QUANTITATIVE RESULTS OF THE MODELS ON THE TRAIN, VALIDATION AND TEST SET.

| Model | Train Loss | Val Loss | Val Accuracy | Test F1-Score | Epochs |
|---|---|---|---|---|---|
| Guessing Baseline | - | - | - | 0.14105 | - |
| Custom CNN | 0.2897 | 0.3034 | 0.9042 | 0.92695 | 64 |
| Pre-trained CNN | **0.0532** | **0.1467** | **0.9537** | 0.96095 | 30 |
| Pre-trained ViT | 0.1438 | 0.2089 | 0.9379 | **0.96725** | 29 |
| **Ensemble** | - | - | - | **0.97103** | - |

Table 2 shows the performance of each model on the validation and the test set. The *custom CNN* achieved a validation accuracy of **90.42%** and a test F1-score of **0.92695**. The *pre-trained CNN* (ResNet-18) outperformed the custom CNN with a validation accuracy of **95.37%** and a test F1-score of **0.96095**. The *pre-trained ViT* (vit-base-patch16-224) achieved a validation accuracy of **93.79%** and a test F1-score of **0.96725**. The *ensemble*, which combined the predictions of all three models, achieved the highest test F1-score of **0.97103**. Since the *guessing baseline* and the *ensemble* are not trained models, the training and validation losses are not applicable.

### 5.2. Qualitative Results

Since the labels for the test set are not available, the qualitative results are based on the validation set to get an idea of the performance of the models. The confusion matrices of the custom CNN, pre-trained CNN and ViT models on the validation set are shown below:

Fig. 11 shows the confusion matrix of the custom CNN on the validation set. The rows represent the true classes, while the columns represent the predicted classes. The diagonal elements represent the number of correct predictions for each class, while the off-diagonal elements represent the misclassifications. While there are some misclassifications without a clear pattern, the model clearly struggled to distinguish between “Loose Silky-bent” and “Black-grass”.

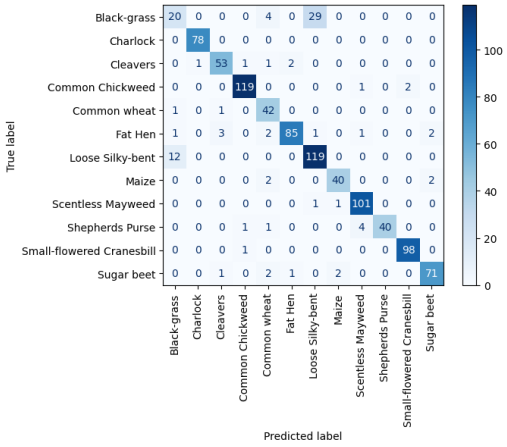



Figure 11. Confusion matrix (custom CNN)

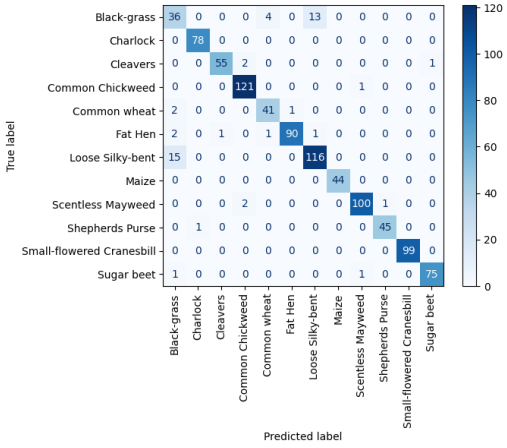



Figure 12. Confusion matrix (pre-trained CNN)

Similar to the custom CNN, Fig. 12 shows the confusion matrix of the pre-trained CNN on the validation set. The model showed the same difficulty in distinguishing between “Loose Silky-bent” and “Black-grass”.

Although the pre-trained ViT model takes a different approach, Fig. 13 shows that it also struggled with the same pair of classes “Loose Silky-bent” and “Black-grass”.

### 5.3. Comparative Analysis

The results show that the ensemble model outperforms the individual models, achieving the highest test F1-score of **0.97103**. The pre-trained ViT model achieved the highest individual test F1-score of **0.96725**, followed by the pre-trained CNN with a test F1-score of **0.96095**. The custom CNN achieved a test F1-score of **0.92695**, demonstrating competitive performance despite being trained from scratch.

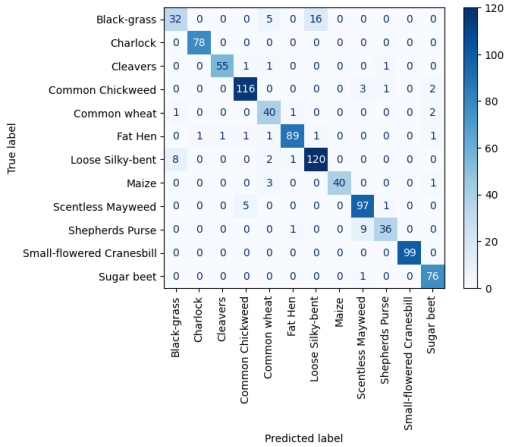



Figure 13. Confusion matrix (pre-trained ViT)

```python
from pytorch_grad_cam import GradCAM
from pytorch_grad_cam.utils.image
    import show_cam_on_image
from pytorch_grad_cam.utils.model_targets
    import ClassifierOutputTarget

with GradCAM(
        model=model,
        target_layers=target_layers,
     ) as cam:
    gs_cam = cam(
                input_tensor
                    =image.unsqueeze(0),
                targets
                    =targets,
             )
    gs_cam = gs_cam[0, :]
    visualization = show_cam_on_image(
                        rgb_img,
                        gs_cam,
                        use_rgb=True,
                    )
```

Listing 7. Generate CAMs using Pytorch-Grad-CAM.

Since real-time inference is not required for this task, the computational cost of the models is not a primary concern. However, the custom CNN is the lightest model with approximately *2 million* parameters, making it computationally efficient. The pre-trained CNN has approximately *11 million* parameters, while the pre-trained ViT has approximately *86 million* (*9 thousand trainable*) parameters, making it the most computationally expensive model.

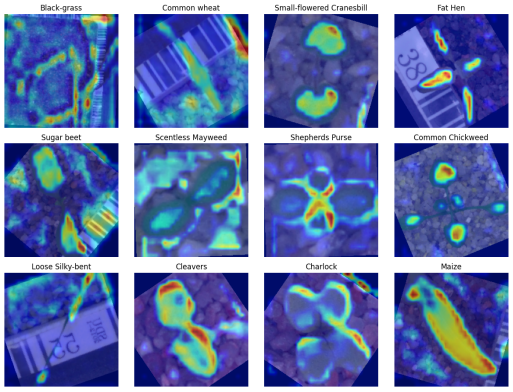



Figure 14. Grad-CAM (custom CNN)

## 5.4. Interpretability Measures

The Pytorch-Grad-CAM library [21] by Jacob Gildenblat was used to generate class activation maps (CAMs) for the custom CNN, pre-trained CNN and ViT models. CAMs provide insight into the regions of the image that the model focuses on when making predictions and can help explain the decision-making process of the model.

The simplified code snippet below shows how to generate CAMs for a given image using the custom CNN model:

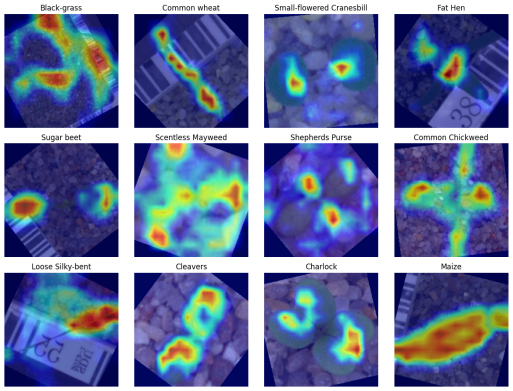



Figure 15. Grad-CAM (pre-trained CNN)

Fig. 14, Fig. 15 and Fig. 16 show the Grad-CAM visualizations for the last two layers of the custom CNN, pre-

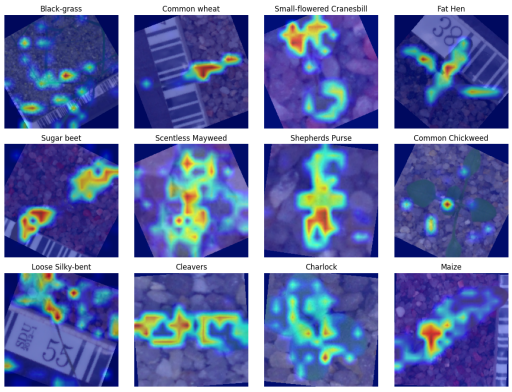



Figure 16. Grad-CAM (pre-trained ViT)

trained CNN and ViT models. The visualizations highlight the regions of the image that the model focuses on when making predictions. The Grad-CAM visualizations provide insight into the decision-making process of the models and help to interpret their predictions.

For some classes, such as “Small-flowered Cranesbill”, “Fat Hen”, “Common Chickweed”, “Cleavers” and “Maize”, the custom CNN clearly focused on parts of the plants that a human would use to distinguish between the classes. For these species, the focus was on the leaves. For classes like “Black-grass”, “Common wheat”, “Sugar beet”, “Scentsless Mayweed” and “Loose Silky-bent” the CNN focused on what appears to be the soil or the background. One could argue that the model had difficulty distiguishing between the plants and the background and focused on *noise* in the images.

In comparison, both pre-trained architectures, the CNN and the ViT, focused more on the plants themselves. But even for “Black-grass”, “Scentsless Mayweed” and “Loose Silky-bent” the models did not seem to focus on the plants alone. The visualizations of the areas of interest are smoother for the CNN compared to the ViT, which looks more *blocky*. This is due to the different architectures and the way the models process the images as the ViT divides the image into patches and processes them separately, therefore the patches are more visible in the Grad-CAM visualizations for the ViT. For a human observer the Grad-CAM visualizations can help understand how the models made their predictions and what features they focused on, the pre-trained CNN seemed to produce the most reasonable visualizations and focus on understandable features.

## 6. Conclusion and Lessons Learned

This section summarizes the key findings and lessons learned from the project. It discusses key takeaways, challenges encountered, a comparison to other challenge submissions and papers and potential future work to improve the results.

### 6.1. Key Takeaways

In this project, several strategies were explored to classify plant seedlings into *12* different species, with the overall goal of achieving robust performance as measured by the mean (micro-averaged) F1-score. Data augmentation played a central role in preventing overfitting and improving model performance. Techniques such as random rotations, flips and color jittering effectively increased the diversity of the training samples, thereby improving the robustness of the learned feature representations. Meanwhile, the choice of an appropriate model architecture proved critical. A custom CNN designed and trained from scratch achieved competitive results (test F1-score of **0.92695**), demonstrating the potential for custom solutions even with relatively modest dataset sizes. However, the use of pre-trained networks, such as `ResNet18` and `vit-base-patch16-224`, demonstrated how transfer learning can deliver superior results (test F1-scores of **0.96095** and **0.96725**) by building on rich feature embeddings learned from large-scale datasets. Proper validation underlined these successes, with a stratified split ensuring balanced class distributions in both the training and validation sets. This practice not only prevented the model from overfitting to majority classes, but also allowed careful monitoring of loss and accuracy metrics to guide training decisions and allowed early stopping to load the best model state before overfitting occurred.

### 6.2. Challenges Encountered

Despite the encouraging results, several challenges remained throughout the process. The class imbalance present in the dataset underscores the need for robust strategies to deal with skewed data, such as stratified data splits. In addition, certain class pairs, such as “Loose Silky-bent” and “Black-grass”, exhibited high visual similarity, leading to consistent confusion for all the custom and pre-trained models. Overfitting remained a significant risk due to the limited dataset size, necessitating the use of multiple regularization methods including weight decay, dropout layers and data augmentation to ensure generalization. Computational constraints also played a role in decisions regarding batch size, image resolution and the capacity of architectures that could be feasibly trained within the available resources (e.g., freezing layers in the ViT model to reduce trainable parameters). Finally, the unavailability of labels for the test set made it difficult to comprehensively evaluate the models, necessitating the use of the validation set as a proxy for performance on unseen data.

### 6.3. Kaggle Challenge Comparison

The Kaggle challenge provided a valuable benchmark to contextualize the performance described. Although the official competition ended in March, 2018, making further submissions and rankings comparisons impossible, analysis of the historical results still provides valuable insights.

The ensemble model achieved a final F1-score of **0.97103** on the test set, placing it approximately at rank *327* out of *833* public participants. Notably, the competition was highly competitive, with the top two submissions achieving perfect scores of 1.0. Unfortunately, the detailed methodologies and codebases of these top-ranked submissions are not publicly available, limiting direct comparison.

However, several published solutions provide useful reference points. The best publicly documented model used the InceptionResNetV2 architecture [22] and achieved an score of 0.98740, closely followed by another highly successful solution based on EfficientNetB0 [23]. In addition, [24] reported impressive results (99.69% accuracy) by combining AlexNet [25] with transfer learning, along with extensive preprocessing steps such as color space conversion and image enhancement. Another study [26] demonstrated strong performance (97.54%) with the VGG19 network [27], outperforming ResNet models in their specific experiments.

This paper is closely aligned with these findings, highlighting the effectiveness of transfer learning for image classification tasks, especially when faced with dataset limitations. However, the uniqueness of this work lies in the explicit investigation of a spectrum of techniques ranging from custom architectures to advanced transfer models such as ViTs, all trained locally and the custom CNN even trained from scratch, while achieving competitive results.

### 6.4. Future Work

Going forward, there are several ways to refine and extend the current results. Adding more models to the ensemble or training the ensemble on the validation set data to find the optimal weights for each model. Exploring deeper pre-trained networks such as ResNet-50, DenseNet, or EfficientNet could improve performance, although careful management of overfitting will be important. Collecting additional labeled data or generating synthetic samples using generative adversarial networks (GANs) [28] could help address the class imbalance and improve the ability of the model to generalize to underrepresented classes. Finally, and probably the best next step, is image segmentation to remove the background "noise" and focus on the plant seedling itself. This could help the models learn more relevant features and improve classification accuracy.

## References


[1] inversion, "Plant seedlings classification," https://kaggle.com/competitions/plant-seedlings-classification, 2017, kaggle.

[2] V. Meshram, K. Patil, V. Meshram, D. Hanchate, and S. Ramkteke, "Machine learning in agriculture domain: A state-of-art survey," *Artificial Intelligence in the Life Sciences*, vol. 1, p. 100010, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2667318521000106

[3] X. Jin, J. Che, and Y. Chen, "Weed identification using deep learning and image processing in vegetable plantation," *IEEE Access*, vol. 9, pp. 10 940–10 950, 2021.

[4] T. M. Giselsson, R. N. Jørgensen, P. K. Jensen, M. Dyrmann, and H. S. Midtiby, "A public image database for benchmark of plant seedling classification algorithms," *CoRR*, vol. abs/1711.05458, 2017. [Online]. Available: http://arxiv.org/abs/1711.05458

[5] inversion, "Plant seedlings classification," www.kaggle.com/competitions/plant-seedlings-classification/overview/evaluation, 2017, kaggle.

[6] C. D. Manning, *An introduction to information retrieval*. Cambridge University Press, 2009.

[7] M. Sokolova and G. Lapalme, "A systematic analysis of performance measures for classification tasks," *Information processing & management*, vol. 45, no. 4, pp. 427–437, 2009.

[8] K. Takahashi, K. Yamamoto, A. Kuchiba, and T. Koyama, "Confidence interval for micro-averaged f 1 and macro-averaged f 1 scores," *Applied Intelligence*, vol. 52, no. 5, pp. 4961–4972, 2022.

[9] C. M. Bishop and H. Bishop, *Deep Learning: Foundations and Concepts*, 1st ed. Springer, 2024. [Online]. Available: https://doi.org/10.1007/978-3-031-45468-4

[10] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016. [Online]. Available: https://www.deeplearningbook.org/

[11] K. O'shea and R. Nash, "An introduction to convolutional neural networks," *arXiv preprint arXiv:1511.08458*, 2015.

[12] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, "An image is worth 16x16 words: Transformers for image recognition at scale," *CoRR*, vol. abs/2010.11929, 2020. [Online]. Available: https://arxiv.org/abs/2010.11929

[13] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *CoRR*, vol. abs/1512.03385, 2015. [Online]. Available: http://arxiv.org/abs/1512.03385

[14] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, 2009, pp. 248–255.

[15] T. Ridnik, E. Ben-Baruch, A. Noy, and L. Zelnik-Manor, "Imagenet-21k pretraining for the masses," *arXiv preprint arXiv:2104.10972*, 2021.

[16] A. Steiner, A. Kolesnikov, X. Zhai, R. Wightman, J. Uszkoreit, and L. Beyer, "How to train your vit? data, augmentation, and regularization in vision transformers," *arXiv preprint arXiv:2106.10270*, 2021.

[17] R. Wightman, "PyTorch Image Models." [Online]. Available: https://github.com/huggingface/pytorch-image-models

[18] J. Stallkamp, M. Schlipsing, J. Salmen, and C. Igel, "Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition," *Neural Networks*, 2012. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0893608012000457

[19] Géron, Aurélien, *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: concepts, tools, and techniques to build intelligent systems*, 3rd ed. O'Reilly, 2022.

[20] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2017. [Online]. Available: https://arxiv.org/abs/1412.6980

[21] J. Gildenblat and contributors, "Pytorch library for cam methods," https://github.com/jacobgil/pytorch-grad-cam, 2021.

[22] C. Szegedy, S. Ioffe, and V. Vanhoucke, "Inception-v4, inception-resnet and the impact of residual connections on learning," *CoRR*, vol. abs/1602.07261, 2016. [Online]. Available: http://arxiv.org/abs/1602.07261

[23] M. Tan and Q. V. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," *CoRR*, vol. abs/1905.11946, 2019. [Online]. Available: http://arxiv.org/abs/1905.11946

[24] C. R. Alimboyong, A. A. Hernandez, and R. P. Medina, "Classification of plant seedling images using deep learning," in *TENCON 2018 - 2018 IEEE Region 10 Conference*, 2018, pp. 1839–1844.

[25] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems*, F. Pereira, C. Burges, L. Bottou, and K. Weinberger, Eds., vol. 25, 2012. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf

[26] E. Hassan, M. Shams, N. A. Hikal, and S. Elmougy, "Plant seedlings classification using transfer learning," in *2021 International Conference on Electronic Engineering (ICEEM)*. IEEE, 2021, pp. 1–7.

[27] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2015. [Online]. Available: https://arxiv.org/abs/1409.1556

[28] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial networks," 2014. [Online]. Available: https://arxiv.org/abs/1406.2661