Classification of 5G base stations

Github link: https://github.com/dmitrii-semenov/ML_project

Team Placeholder: Dmitrii Semenov, Josef Pecka

03.05.2025

1. Introduction

This project addresses the classification of such base stations using machine learning techniques applied to channel frequency response (CFR) data.

The objective is to build a classifier capable of identifying three classes:

- Class 0: Legitimate eNodeB (5G base station)
- Class 1: Attacker at location 1
- Class 2: Attacker at location 2

Using CFR samples and supervised learning, we aim to distinguish between these three classes with high accuracy (above 99%).

2. Problem Description

Each CFR sample is a 2D matrix with dimensions 72 (subcarriers) by 48 (repetitions). The training dataset contains labeled examples, while the test set is used for final evaluation through Kaggle. The core challenge is correctly classifying new, unseen data into one of the three predefined classes.

We have also noticed that the dataset is imbalanced, which means that class 0 is the dominant (around 80-90%) part of the dataset. Therefore, we decided to include a data augmentation block, particularly focusing on restoring the correct label distribution of the input data.

3. Data preprocessing

The whole architecture of our code is presented in the block diagram in **Fig.1**.

We started by loading all training ".npy" files based on the filenames listed in "label_train.csv." Then, we normalized the data using Z-score normalization, which helps stabilize training by ensuring the input has zero mean and unit variance.

Next, we split the dataset into training and validation sets using the "train_test_split". Here, we used a classical 80 to 20 split, which was actively used during the PC labs this semester.

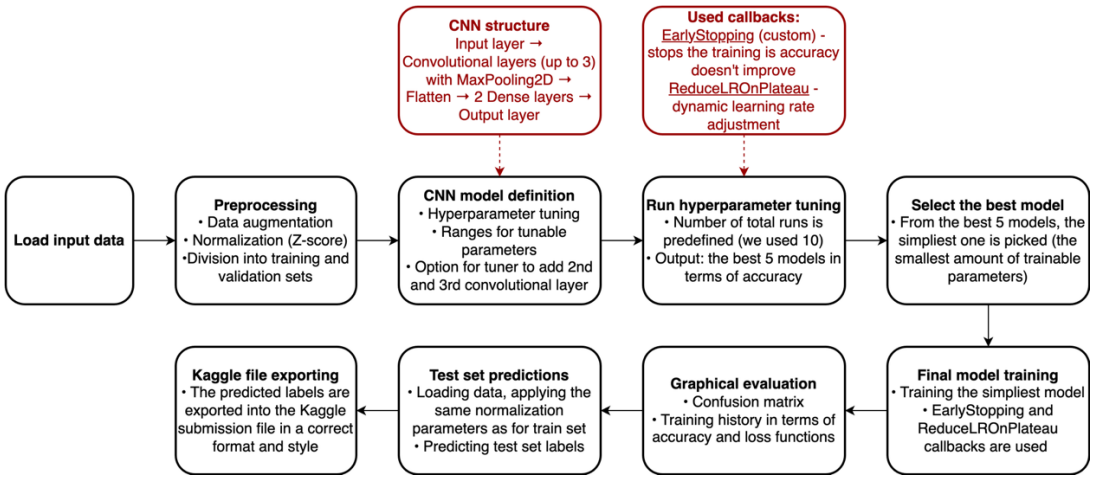

Fig. 1 Complete Block diagram of the presented solution

At this point, we understood that the dataset was imbalanced - class 0 (legit base stations) was much more frequent than classes 1 and 2 (attackers). To fix this, we implemented a custom data augmentation function. It adds Gaussian noise to underrepresented classes, especially classes 1 and 2, so all classes end up with similar counts. This was done before splitting the data to ensure the train and validation data have the same distribution of classes. The final distribution after augmentation and data splitting is presented in **Fig.2**.

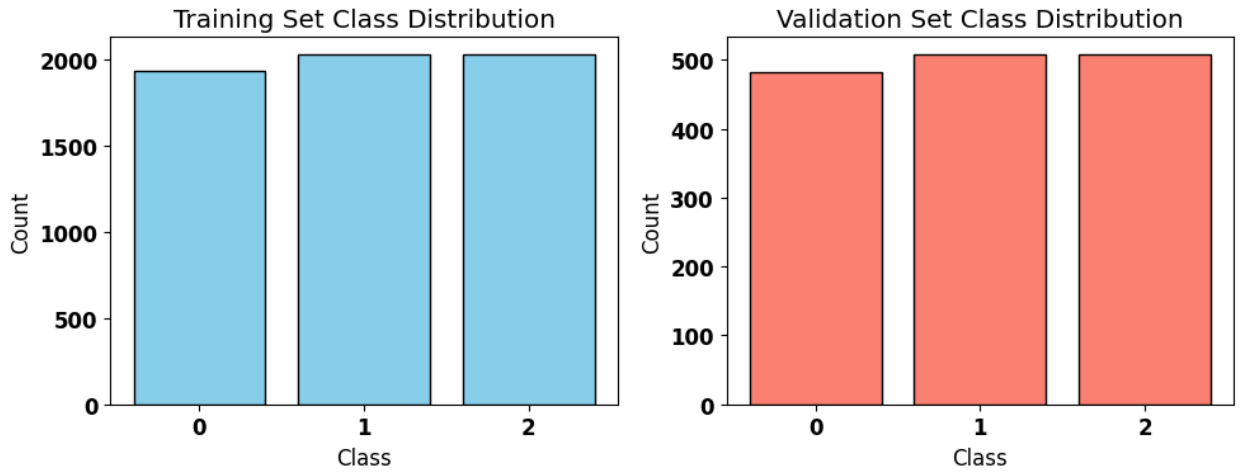

Fig. 2 Class distribution in the training and validation sets after augmentation

4. Model architecture

We used a classical 2D Convolutional Neural Network (CNN) to solve the task. The model contains:

- One convolutional layer (Conv2D) with tunable filters and kernel size

- An optional 2nd and 3rd convolutional layer (controlled via boolean variables)
- MaxPooling2D after each convolutional layer
- Flatten, 2 Dense layers (with tunable size)
- Dropout and final Sense layer with softmax activation

We also added a custom callback "EarlyStoppingWithThreshold", which works as a classical "EarlyStopping", but starts to monitor the required metrics only after it reaches a pre-defined threshold. The motivation behind it is that in some cases (mostly for complex models), it takes up to 10 epochs before the model starts to train (so that the "val_accuracy" increases and "val_loss" decreases). The custom callback was developed to wait over this period and prevent the training from stopping at this stage. One more callback we used is "ReduceLROnPlateau", which decreases the learning rate dynamically, if needed, and can help to achieve a high accuracy faster (in fewer epochs).

Therefore, the model is very flexible in terms of options, and by using a hyperparameter tuner, it is possible to find the simplest model out of the best-performing models.

We used Keras Tuner with the "RandomSearch" option to get the best configuration. Here, the tuner searches through a pre-defined number of random combinations (at the end, we set it to 10 combinations as a compromise between execution time and dispersion of models). The parameters we tuned include:

- Number of filters: 64, 128, 256, 512
- Kernel size: 3 or 5
- Size of the dense layer: from 64 to 512
- Dropout rate: from 0.2 to 0.5
- Learning rate: between 0.0001 and 0.01
- Boolean variables to include 2nd and 3rd convolutional layers

From the top 5 models with the highest validation accuracy, we picked the one with the smallest number of trainable parameters, as seen in **Fig.3**. The further training metrics for the best model are presented in **Fig.4**.

```
Model 1 - Total trainable parameters: 1334403
Model 2 - Total trainable parameters: 759747
Model 3 - Total trainable parameters: 759619
Model 4 - Total trainable parameters: 233155
Model 5 - Total trainable parameters: 333379
Picked model 4
```

Fig. 3 The best model picked out of the five top-performing

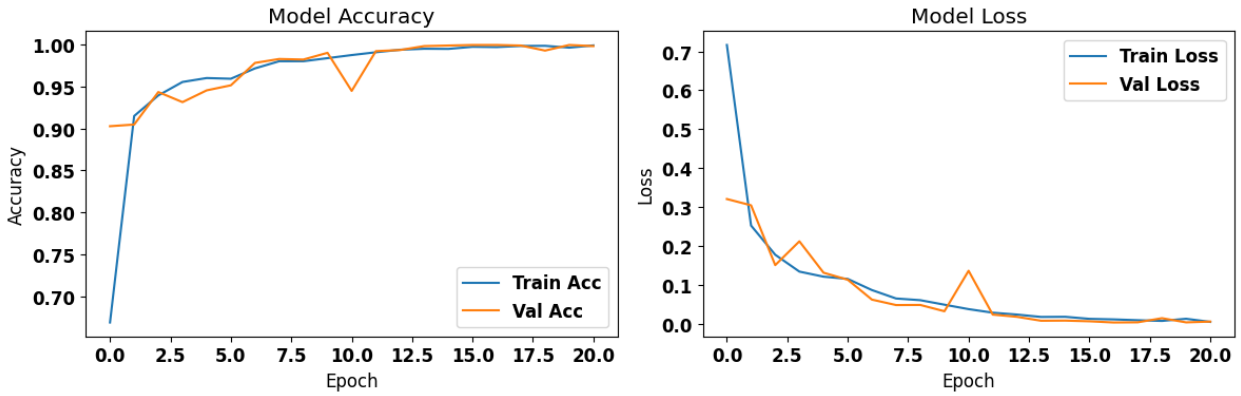

Fig. 4 Accuracy and loss graphs for the selected model

5. Results evaluation

After training and tuning, our model achieved 99.8% accuracy. The detailed results with misclassified detection are presented in the Confusion matrix in **Fig.5**. After predicting labels for test data and submitting it to Kaggle, the actual score is 0.99166, which is the top score at the moment and shows that the designed model is capable of predicting labels correctly in almost all cases.

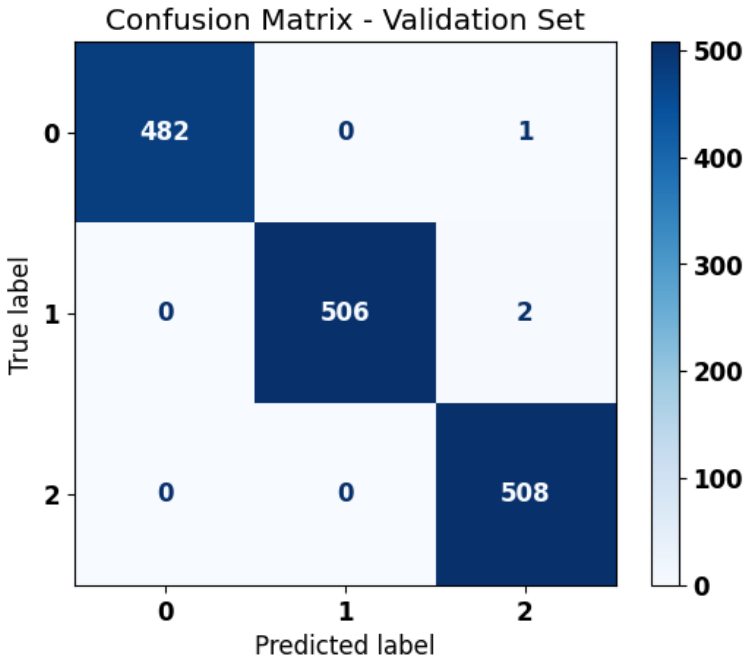

Fig. 5 Confusion matrix of the best model

6. Conclusion

This project focused on developing a neural network for classifying 5G signal stations across three classes. As a preprocessing step, Z-score normalization was applied to standardize the input feature distributions. Due to a significant class imbalance in the original dataset, we employed data augmentation by adding Gaussian noise to samples from the minor classes. This strategy effectively equalized the number of samples per class while keeping features for training.

The best training model was selected using an automated hyperparameter tuning, which evaluated multiple candidate models. The final model was selected based on a trade-off between performance and computational efficiency, selecting the architecture with the lowest number of trainable parameters to minimize memory footprint and storage requirements. Despite this memory optimization, the chosen model achieved a classification accuracy of 99.166% on the test dataset, correctly predicting 119 out of 120 samples. These results demonstrate that even with memory optimization, achieving one of the best results across the whole Kaggle competition is possible.

7. References

1. Chollet, F. (2015). *Keras: Deep learning for humans*. https://keras.io — Accessed: May 3, 2025.
2. Pedregosa, F., et. al. (2011). *Scikit-learn: Machine learning in Python*. Journal of Machine Learning Research, 12, 2825–2830. https://scikit-learn.org — Accessed: May 3, 2025.
3. Kaggle. (2025). *MLF_MPA_2025 Final Project Competition Page*. https://www.kaggle.com — Accessed: May 3, 2025.
4. TensorFlow Addons. (2024). *Focal loss implementation for classification*. GitHub Repository. https://github.com/tensorflow/addons — Accessed: May 3, 2025.
5. Brno University of Technology. (2025). *MLF_MPA_2025 Course Materials and Lectures*. Internal Moodle Platform — Accessed: May 3, 2025.