Classification of false 5G base stations using PSS and SSS sequnces

Václav Kubeš
Department of Radio Electronics
FEEC, Brno University of Technology
Czech Republic
240644@vutbr.cz

Hana Daová
Department of Radio Electronics
FEEC, Brno University of Technology
Czech Republic
230238@vutbr.cz

***Abstract*—In modern cellular networks, the proliferation of rogue base stations (RBS), also known as false base stations (FBS), poses a significant security threat. These devices mimic legitimate 4G/LTE base stations to deceive mobile users and potentially intercept sensitive data. This project explores the classification of base stations through machine learning, utilizing the channel frequency response of synchronization signals (PSS/SSS). We propose a simple, yet effective convolutional neural network (CNN) model trained to distinguish between legitimate and false base stations across three classes. To enhance the robustness of the model and improve classification accuracy, additive white Gaussian noise (AWGN) was applied to augment the training data set. The combination of CNN architecture and data augmentation yielded strong performance, demonstrating the model's capability to detect RBS activity in this particular scenario.**

***Index Terms*—Convolutional neural network, Classification, Base station, Synchronization sequence**

I. Introduction

Mobile devices as we know them today have become widely spread quite recently, causing a great demand for mobile internet connection. This demand is satisfied by mobile communication systems such as 4G and 5G. The wireless connection to user equipment (UE) is mediated by base stations (BTS) called Evolved NodeB (eNB) in 4G and Next Generation NodeB (gNB) in 5G [1]. To ensure synchronization, the base station transmits synchronization sequences. The primary synchronization sequence (PSS), which boosts subframe synchronization, is sent right after UE activation and its connection to the communication network. Once PSS is successfully detected, the entire synchronization signal block (SSB), which also consists of a secondary synchronization sequence (SSS), is decoded. In other words, the sequences are used to obtain physical layer cell identities or to detect radio frame timing. Both PSS and SSS are transmitted on all 72 subcarriers of OFMD.

One of the most well-known attacks on the security of mobile communication networks such as 5G is performed by false base stations. FBS are unauthorized cell towers that attack mobile devices (UEs) and networks. They exploit UEs' tendency to connect to the strongest signal by broadcasting powerful, often mimicked, signals to force connections. A major attack using these signals is the IMSI-Catcher, which intercepts unprotected subscriber identities (IMSIs). This compromises privacy and allows attackers to track user locations or mimic the user's identity [2].

During a measurement campaign, multiple PSS and SSS were collected from one legitimate (BTS0) and two false base stations (BTS1 and BTS2) and they were used to calculate the channel frequency response [3]. These calculated frequency responses were then stored in a dataset, which will be used to train our neural network to distinguish between legitimate and false base stations. One sample is represented as a 72 (subcarriers) × 48 (repetitions of channel frequency responses) matrix of values and can be depicted as 1. The initial number of samples of legitimate BTS is 1209 and the number of each false BTS is 141.

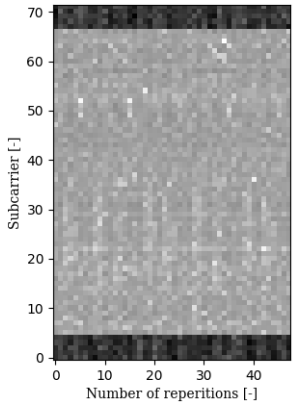

Fig. 1. Depicted channel freq. response determined from PSS/SSS.

II. Neural network selection

Selection of the suitable neural network which would classify given channel frequency responses is crucial for achieving the best performance and precision. There are two types of neural network that should be considered, the convolutional neural network (CNN) and the multi-layer perceptron (MLP). Since the dataset consists of matrices that can be interpreted as black and white images, CNN seems to be the right choice. The convolutional layers can efficiently extract the key features needed for accurate classification, which is the main reason why they are widely used for image recognition tasks.

III. Dataset augmentation and preprocessing

Due to the insufficient number of samples simulating false base stations, data augmentation becomes a necessary step. To augment the dataset, false BTS data is replicated with added AWGN noise. The standard deviation of the noise can be varied as the number of data repetitions increases, which also increases the robustness and generalization of the model trained on the dataset. To increase the accuracy and generalization even further, the number of legitimate base station is augmented in the same way, leading to a total number of 2892 BTS0, 2892 BTS1 and 2913 BTS2. A snippet of the code used for the augmentation of the BTS1 class data can be seen below 1.

```
for k in range(num_of_bts0//bts_1.shape[0]):
    bts_1 = bts_1 + np.random.normal(0,\
        awgn_std + k//3, (bts_1.shape[0],\
        bts_1.shape[1], bts_1.shape[2]))

    dataset = np.append(dataset, bts_1, 0)

    dataset_labels = np.append(dataset_labels,\
        np.ones(bts_1.shape[0]))
```

Listing 1. Data augmentation

For the best training outcome, one-hot encoding is applied to the classification labels. Standardization is used within the augmented dataset to decrease the amplitude of the input values.

IV. Convolutional neural network architecture

As mentioned above, the input data samples are shaped into 72 (number of subcarriers allocated to PSS/SSS) × 48 (repetitions of channel frequency responses) matrices with a single channel (B&W image). Thus, the first layer of the network, serving as the input layer, is configured to accept grayscale images of dimensions 72×48×1. This layer defines the structure of the incoming data and ensures compatibility with the subsequent layers.

The data are then processed by a two-dimensional convolutional layer (Conv2D) employing a certain number of filters with a symmetrical kernel size and the ReLU (Rectified Linear Unit) activation function. The primary role of this layer is to extract simple patterns from the input images. The use of ReLU introduces non-linearity into the model, enabling it to learn complex feature representations.

Following convolution, a pooling layer with a symmetrical pool size is applied. This layer performs downsampling by selecting the maximum value from each non-overlapping region of the feature maps, thereby reducing their spatial dimensions. Pooling serves to decrease the computational load and control overfitting.

Subsequently, a Flatten layer converts the two-dimensional feature maps into a one-dimensional feature vector. This operation is necessary for transition from the convolutional part of the network to the fully connected part, enabling the model to combine local features into global interpretations.

The flattened feature vector is then passed through the first Dense (fully connected) layer with a specified number of neurons and ReLU activation. This is followed by a second Dense layer, featuring fewer neurons and the same activation function, which further condenses and refines the learned representations.

The architecture ends with a Dense output layer containing three neurons representing each class and a Softmax activation function. This function generates a probability distribution over the target classes, enabling the network to perform multi-class classification. The output class corresponds to the neuron with the highest probability score. The initial base architecture is depicted in figure 2.

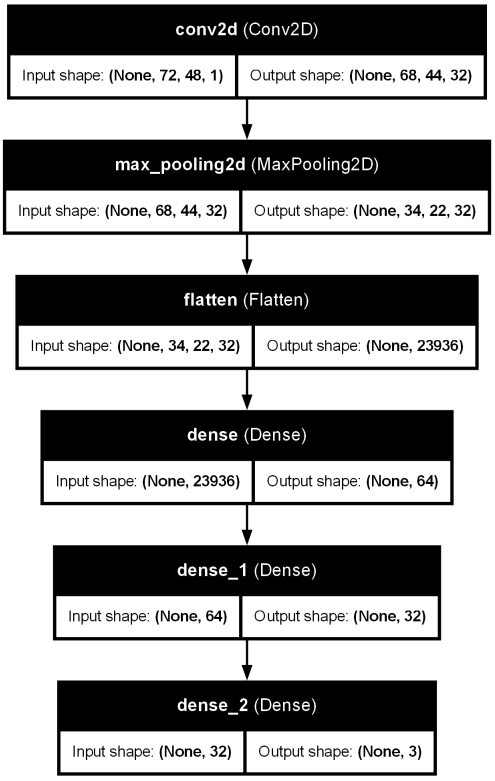

Fig. 2. Initial base architecture of the CNN.

V. Hyperparameter Tuning

Given the simplicity of our model, it is beneficial to implement an automated hyperparameter tuning mechanism. By adjusting various combinations of hyperparameters such as batch size, number of epochs, regularization, learning rate, number and type of layers, optimizers, number of neurons, kernel size, number of filters, pool size, a stochastic semi-automatic tuning process was developed.

Based on manual testing, an additional convolutional layer with half the number of filters used in the first convolutional layer and ReLU activation followed by another pooling layer, had been added to our model's base architecture to improve CNN performance.

In the context of the convolutional layers, values from predefined lists representing either number of filters or kernel size are processed in a for loop. Increasing the number of values and the values themselves generally raises computational demands, and in our case, we chose five values (16, 24, 32, 48, 64) for number of filters and three dimensions (2×2, 5×5, 8×8) for kernel size. When testing different kernel sizes, larger ones caused excessive smoothing and a loss of characteristic features, while smaller ones failed to capture enough relevant information. Comparisons of losses of different kernel sizes and numbers of filters tested with only one convolutional layer for illustrative purposes are depicted in fig. 4 and fig. 3 respectively. Each change in the hyperparameters was made without any change to others and with preservation of the initial base architecture.

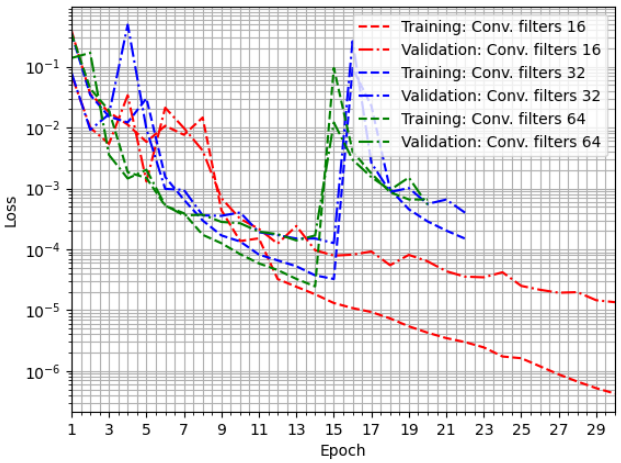

Fig. 3. Loss of the proposed architecture for different number of filters in conv. layer.

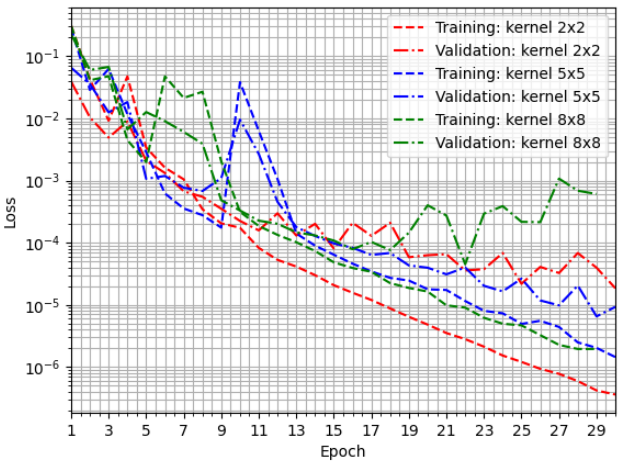

Fig. 4. Loss of the proposed architecture for different kernel sizes in conv. layer.

The pool size in the pooling layer directly affects the spatial dimensions of the data. To preserve the dominant features of the input, a max pooling algorithm was chosen for our CNN. It is advisable to apply a smaller pool size initially, meaning the pool size of the first pooling layer should be the smallest (or at least equal to that of the following pooling layers). In a for loop that encloses the loop for convolutional layers, four selected dimensions (2×2, 3×3, 4×4, 5×5) of the pool size are tested. Using a larger pool resulted in too much downsampling, which removed critical information and negatively impacted the model's performance, as can be seen in figure 5.

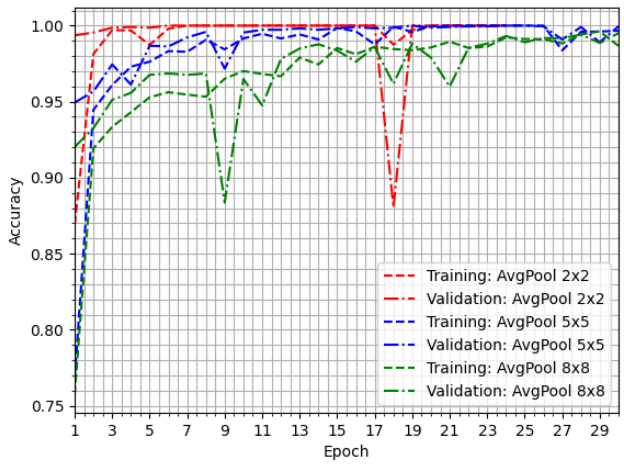

Fig. 5. Accuracy of the proposed architecture for different pool sizes in AvgPool. layer.

After flattening the data from multiple matrices into a single vector, the use of fully connected layers is recommended. The number of layers and their neurons have a major influence on the convergence speed and the accuracy achieved. In our model, two fully connected layers are used, one layer with 48 neurons and another one with 25 neurons. The number of neurons must be set manually and can be changed only at the beginning of the computation.

To make the tuning algorithm stochastic, two Dropout layers are added after each dense layer. The dropout layer randomly disables a specified fraction of neurons making them appear as if they were not present at the first place. The dropout rate is selected randomly for each run with a range from 0 to 0.5.

Another stochastic approach is implemented by conditionally adding BatchNormalization layers based on a randomly selected boolean value each run. This layer stabilizes and accelerates the training process, and it may be inserted before or after the first dropout layer, or after the second dropout layer or not added at all.

As for optimizers, there are many options available in machine learning. For image classification tasks like ours, the most commonly used are Adam and AdamW. In this project, Adam was preferred, as AdamW showed poorer convergence. To demonstrate the effect of using an incompatible optimizer, SGD was also tested and compared with Adam and AdamW on the figure 6. The learning rate of the network was set accordingly to ensure the model could learn effectively without unnecessarily extending the training time.

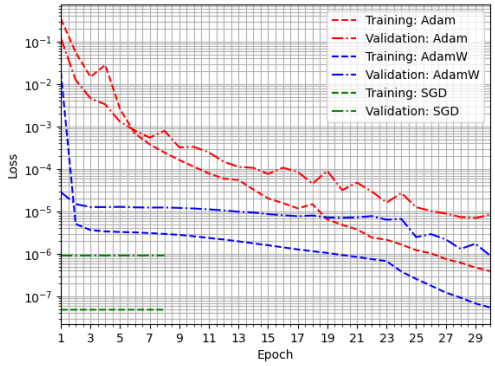

Fig. 6. Loss of the proposed architecture for different optimizers.

Following the optimizer setup, it was necessary to define several other hyperparameters. A batch size of 20 was chosen to allow smooth weight updates, supporting both stable and efficient learning. The number of epochs was set to 30, offering sufficient flexibility for computation. To secure the best weighting, an early stopping callback was implemented, restoring the weights to the point where the validation error was lowest.

Finally, for optimal model evaluation, the F1 score of each compiled network is tracked. F1 represents a combination of precision and recall, and it is suitable for our task, which involves similar classes with subtle distinguishing features. Only the model with the highest score across all runs is saved [4]. In case multiple models achieve the same F1 score, a backup metric is used. The first backup metric is the validation loss, and if that is also equal, validation accuracy is considered.

VI. Results

An example of a model with the best results using autotuning is in the core, the CNN architecture mentioned above with adjusted parameters. The figure 7 shows loss and accuracy of its performance, figure 8 shows confusion matrix after autotuning process completion and figure 9 shows the final model architecture.

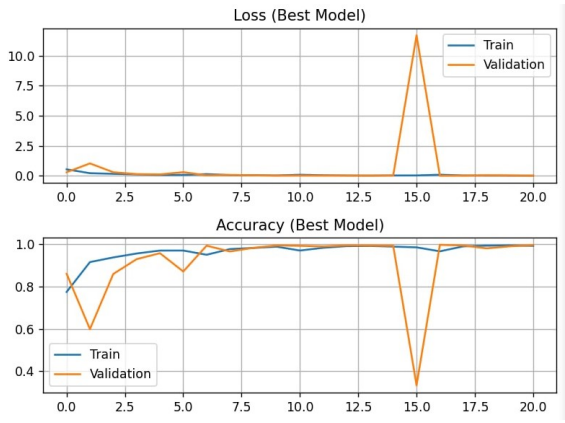

Fig. 7. Loss and Accuracy of the proposed architecture.

An example of the final hyperparameters are listed in table I.

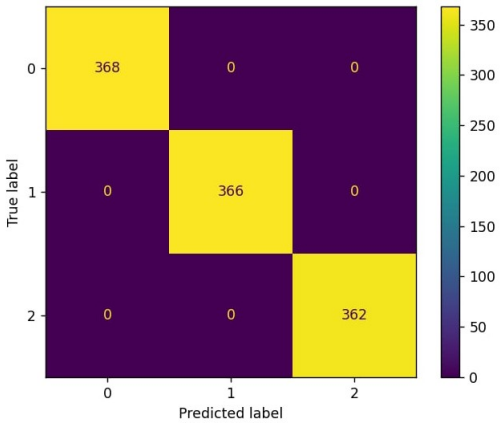

Fig. 8. Confusion matrix of the proposed model.

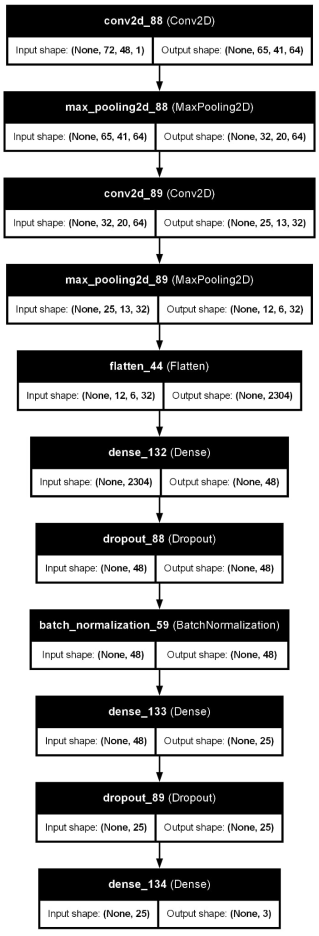

Fig. 9. Example of the final architecture of the CNN.

TABLE I
MODEL HYPERPARAMETERS

Parameter	Value
Optimizer	Adam
Learning rate	0.001
Batch size	20
Overfitting countermeasure	EarlyStopping
Patience	4
Conv. Layer 1 kernel size	(8, 8)
Conv. Layer 1 no. of filters	64
MaxPool 1 pool size	(2, 2)
Conv. Layer 2 kernel size	(8, 8)
Conv. Layer 2 no. of filters	32
MaxPool 2 pool size	(2, 2)
Dropout 1	0.125
Dense Layer 1 no. of neurons	48
Dropout 2	0.103
Dense Layer 2 no. of neurons	25

VII. CONCLUSION

Within this project, a simple convolutional neural network achieving a classification accuracy of up to 99.17 % was developed. The most important factor in reaching this performance was the input data augmentation using AWGN noise. Taking the results into account, the proposed automated hyperparameter tuning algorithm provided a model with satisfactory classification accuracy, but at the cost of increased computational time.

Since our compiled neural network misclassified only 0.83 % of the unseen data, it can be considered reliable for educational purposes. However, it remains questionable whether this level of performance would be sufficient for critical applications as well.

The whole code and used dataset can be accessed on GitHub via this link [5].

REFERENCES

[1] 5g Networks, "5g terminology: The gnb - 5g networks," 5G Networks, 05 2019. [Online]. Available: https://www.5g-networks.net/5g-terminology-the-gnb/

[2] H. Park, P. Virgil, Y. Ko, Y. Park, T. Kim, and I. You, "Smdfbs: Specification-based misbehavior detection for false base stations," *Sensors*, vol. 23, pp. 9504–9504, 11 2023. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC10708569/

[3] R. Marsalek, "Mpa-mlf final project description," 04 2025.

[4] N. Buhl, "F1 score in machine learning," encord.com, 07 2023. [Online]. Available: https://encord.com/blog/f1-score-in-machine-learning/

[5] vaclav-kubes and , hakidaova, "GitHub - vaclav-kubes /MLF_project: MPA-MLF - Final project: Classification 5G base stations," GitHub, 2025. [Online]. Available: https://github.com/vaclav-kubes/MLF_project

[6] OpenAI, "Chatgpt," ChatGPT, 2025. [Online]. Available: https://chatgpt.com/

[7] Group of unknown authors and Spiritus Sanctus, *Vetus Latina: Gn 1, 3*. Vatican, 350 AD.