Sangar THAYAPARAN

Sherine BEN YAALA

MPA-MLF- Final project- ISEP

Study Module : FEEC

Year of study : 2025

Title of the module : Machine Learning Fundamentals (MPA-MLF 24/25L)

1. Introduction

In this project, the goal was to create a model capable of automatically recognizing images classified into 3 different categories each image in a . npy separated, and labels in a .csv. So we had to start by organizing and loading the data.

In this project, we built a deep learning model using convolutional neural networks (NNC) capable of distinguishing the three categories. We have also overcome the imbalance that there was between the classes of the dataset.It was also decided to start from a simple model and not to use a hyperparameter

2. Problem Description

This project is framed as a **supervised image classification task**, where the goal is to develop a machine learning model that can correctly identify the class of a signal based on its visual representation. Each signal has been preprocessed into a fixed-size 2D grayscale image, which makes it suitable for computer vision models like convolutional neural networks (CNNs).

The dataset consists of files in .npy format, which are NumPy arrays representing images with a resolution of **72 rows by 48 columns**. These images are not traditional photographs but rather **channel frequency response matrices**, extracted from LTE/4G synchronization sequences specifically, the PSS (Primary Synchronization Sequence) and SSS (Secondary Synchronization Sequence).

These 72×48 matrices are essentially a visual fingerprint of the radio environment at a given location, which can be used to identify anomalies such as rogue base stations.

Classification part divides in 3 parts:

Class	Description
0	Legitimate base station (T-Mobile)
1	Fake station – Location 1
2	Fake station – Location 2

Table 1: summary of classes

3. Methodology

To complete this image classification project, we followed a standard deep learning pipeline, step by step: data preparation, model building, training, and evaluation.

3.1 Data Loading

The training images were provided as *.npy* files stored in a folder called Train. Each file contains a single grayscale image. The labels for these images were listed in a separate file named *label_train.csv*, which matched each image ID to its class (0, 1, or 2).

We created a custom Python function to:

- Read the CSV file using pandas libraries
- Match each ID to its corresponding *.npy* file
- Load the image data using numpy libraries and store it in a list
- Store the associated label in a separate list

For the test set who used for submission, we simply loaded all *.npy* files from the Test folder in sorted order.

3.2 Preprocessing

Preprocessing is a crucial step for preparing the data for input into a CNN model. Several steps were applied after loading the dataset:

1. **Normalization of Pixel Values**:
   The pixel values were scaled from the original range from 0 to 255, we decide to normalize it by dividing by 255. This helps the model train more efficiently
2. **Reshaping Images**:
   We decided to reshape each image from (72, 48) to (72, 48, 1) to include the grayscale channel, making it compatible with the CNN input.
3. **Encoding Labels**:
   Since we're dealing with 3 classes, we converted the labels by using *to_categorical()*. This makes the labels usable with the softmax output and categorical crossentropy loss.
4. **Train-Validation Split**:
   We randomly split the dataset into 80% training and 20% validation using *train_test_split()* from scikit-learn. This helped us monitor the model's performance during training.

3.3 CNN Architecture

For the CNN architecture, we used a relatively simple structure suitable for this classification task, which does not involve hyperparameter tuning. The architecture consists of:

1. Conv2D Layer: The first layer is a convolutional layer with 32 filters and a kernel size of (3, 3). It uses the ReLU activation function to introduce non-linearity and learns the spatial features from the images.

2. MaxPooling2D Layer: This layer reduces the spatial dimensions of the image, helping to reduce computational load and extract the most important features
   .

3. Conv2D Layer: A second convolutional layer with 64 filters, again using ReLU activation.

4. MaxPooling2D Layer: Another max-pooling layer to downsample the feature maps.

5. Flatten Layer: This layer converts the 2D feature maps into a 1D vector to feed into the fully connected layers.

6. Dense Layer: The final dense layer has 3 units (one for each class) and uses a softmax activation function to output probabilities for each class.

Model: "sequential_2"

Layer (type)	Output Shape	Param #
conv2d_4 (Conv2D)	(None, 70, 46, 32)	320
max_pooling2d_4 (MaxPooling2D)	(None, 35, 23, 32)	0
conv2d_5 (Conv2D)	(None, 33, 21, 64)	18,496
max_pooling2d_5 (MaxPooling2D)	(None, 16, 10, 64)	0
flatten_2 (Flatten)	(None, 10240)	0
dense_4 (Dense)	(None, 3)	30,723

Total params: 49,539 (193.51 KB)
Trainable params: 49,539 (193.51 KB)
Non-trainable params: 0 (0.00 B)

Image1: result of model.summary

3.4 Compilation & Training

After defining the CNN model, we compiled it by specifying the optimizer, loss function, and evaluation metrics:

- Optimizer: We used the Adam optimizer, which is well-suited for many types of machine learning problems and typically works well without needing much tuning.

- Loss Function: Since this is a multi-class classification problem, we used categorical crossentropy as the loss function.

- Metrics: We chose accuracy as the evaluation metric.

The model was trained for 10 epochs using a batch size of 64. We trained the model using the *.fit()* method, passing in both the training and validation data so we could track the performance on unseen data throughout the training process.

4. Evaluation

4.1 Confusion Matrix

We used the **confusion matrix** to further evaluate the performance of the model across the three classes. The confusion matrix is a useful tool for understanding how well the model is performing on each individual class and where it might be making errors.

- **Class 0 (Legitimate station)**: 237 out of 239 samples were correctly classified. This strong performance is due to the large number of class 0 examples in the training set, which helped the model learn its patterns effectively.
- **Class 1 (Fake station – Location 1)**: 25 out of 30 were correctly predicted, while 5 were misclassified as class 2. This confusion suggests that the model had difficulty separating the two fake station classes, likely because of similar patterns and fewer training samples.
- **Class 2 (Fake station – Location 2)**: 29 out of 30 were correctly predicted, with only 1 misclassified as class 0. The model handled this class better, though the misclassification again reflects a slight bias toward the dominant class.

The model performs strongly on **class 0**, which has the most training data, but its performance drops on **classes 1 and 2**, which are underrepresented. The limited number of samples for the fake stations makes it harder for the model to generalize and leads to higher misclassification rates for those classes.

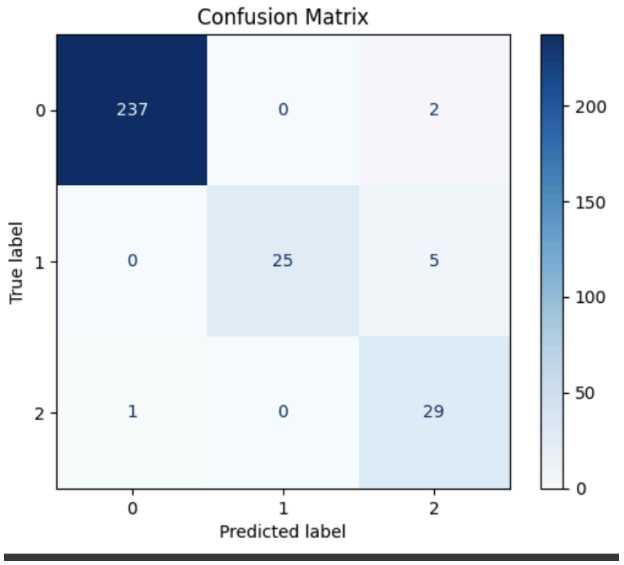

Image2: Confusion Matrix

5. Results and Predictions

After training, we evaluated the model on the validation set and achieved a final accuracy of approximately 96%, which indicates strong generalization despite class imbalance. The model showed excellent performance on the majority class (class 0), while maintaining reasonable accuracy on minority classes.

To generate predictions for the test set, we applied the trained model to the 120 test images. The model outputted probabilities for each class, which were then converted into final class predictions by taking the class with the highest score.

Loss over Epochs:
The training loss consistently decreased, indicating that the model was effectively learning from the data. The validation loss also decreased overall, with some minor fluctuations, which is normal. Importantly, there was no significant gap between training and validation loss, suggesting that **the model did not overfit** and generalized well to unseen data.

Accuracy over Epochs:
The training accuracy steadily increased and reached **~98.7%**, while the validation accuracy improved up to **~97.3%** by the end of training. The close performance between both curves confirms that the model was not memorizing the training data but instead learning patterns that also applied to the validation set.

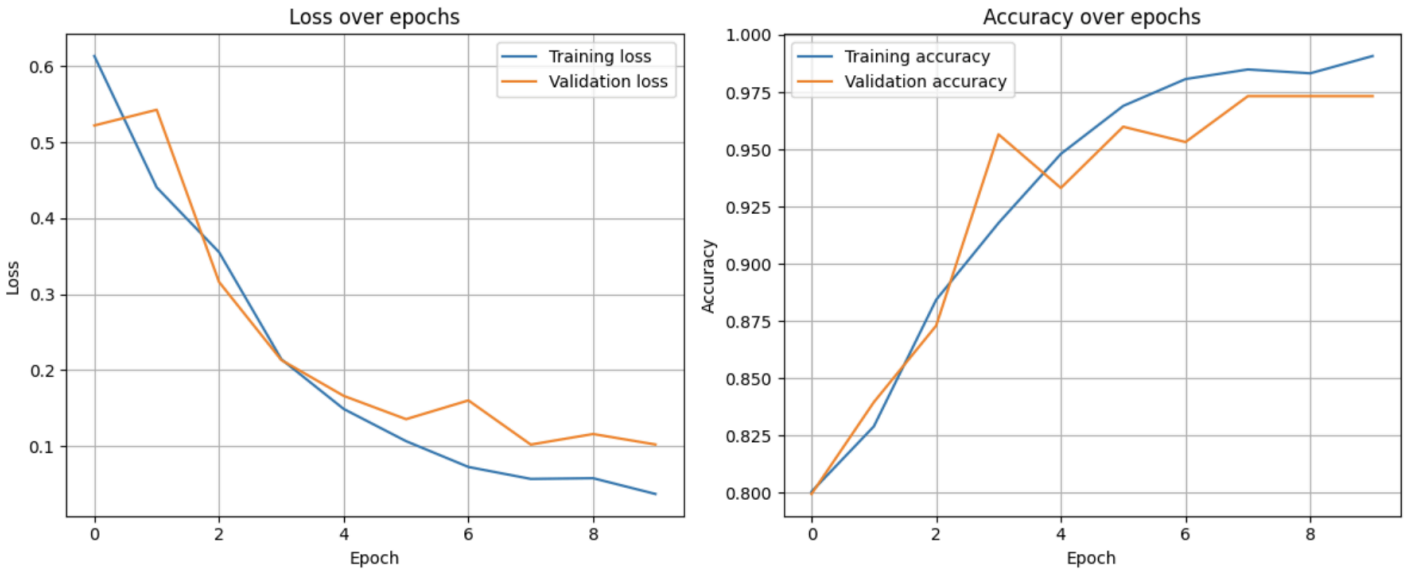

Image3: Evolution of Loss and Accuracy Throughout Training

6. Conclusion

In this project, we developed and trained a convolutional neural network (CNN) model to classify images into three distinct categories. Starting from raw *.npy* image files, we carefully loaded, reshaped, and normalized the data, and prepared the labels for training. Throughout the process, we faced several challenges, such as class imbalance and low validation accuracy, but we managed to overcome them by refining our preprocessing and using a clear, efficient model structure.

Although we intentionally avoided advanced hyperparameter tuning techniques to keep the process simple and focused, the final model still achieved solid performance, especially on the training set. The accuracy and loss curves show a steady improvement over epochs and indicate that the model was learning effectively.

7. References

- TensorFlow Keras Documentation
- Scikit-learn utilities
- Course Labs
- NumPy, pandas libraries
- GITHUB: SherineBY/MPA-MLF--Final-project