

|  |
| --- |
| **Drone Identification Using Micro-Doppler** |

|  |  |
| --- | --- |
| 24-1-1-3174 | **Project Number:** |

|  |
| --- |
| **Project Report** |

|  |  |  |  |
| --- | --- | --- | --- |
| Student: | Amit Stein | ID: | 207195280 |
| Student: | Niv Avivi | ID: | 208731901 |

|  |  |
| --- | --- |
| Supervisor: | Omer Tzdiki |

|  |  |
| --- | --- |
| Project Carried Out at: | Tel Aviv University – Radar Laboratory (Principal Investigator: Prof. Pavel Ginzburg) |

Contents

[Contents 1](#_Toc202695109)

[Abstract 4](#_Toc202695110)

[1 Introduction 6](#_Toc202695111)

[1.1 Motivation 6](#_Toc202695112)

[1.1.1 Challenges in Drone Detection and Classification: 6](#_Toc202695113)

[1.1.2 Solution to Challenges: 6](#_Toc202695114)

[1.2 Project Objective: Accurate Classification of Noisy Micro-Doppler Signals Using CNN…………………………………………………………………………7](#_Toc202695115)

[2 Theoretical background 8](#_Toc202695116)

[2.1 Doppler tags- 8](#_Toc202695117)

[2.2 Spectrogram- 8](#_Toc202695118)

[2.3 SNR 9](#_Toc202695119)

[2.4 Synthetic Data 9](#_Toc202695120)

[2.4.1 The Necessity of Synthetic Data for Micro-Doppler Drone Classification: 9](#_Toc202695121)

[2.4.2 Creating Synthetic Data: Techniques and Methods: 9](#_Toc202695122)

[2.4.3 Time Masking technique: 10](#_Toc202695123)

[2.5 CNN Architecture and Hyperparameters 10](#_Toc202695124)

[2.5.1 Architecture …………………………………………………………..10](#_Toc202695125)

[2.5.2 CNN Key Hyperparameters 11](#_Toc202695126)

[2.6 Alternative algorithms for project implementation 13](#_Toc202695127)

[2.6.1 Alternative Algorithm: K-Nearest Neighbors (KNN) 13](#_Toc202695128)

[2.6.2 Noise‑Robust Alternatives Considered 14](#_Toc202695129)

[2.6.3 The CNN Model from the DopplerTag Paper and Our Comparative Approach ……………………………………………………………………..14](#_Toc202695130)

[3 Simulation 16](#_Toc202695131)

[4 Implementation 18](#_Toc202695132)

[4.1 Synthetic Data Pipeline: Preprocessing, Augmentation, and Dataset Splitting…………………………………………………………………………..18](#_Toc202695133)

[4.1.1 Preprocessing and Data Preparation 18](#_Toc202695134)

[4.1.2 Spectrogram Generation and Dataset Splitting 18](#_Toc202695135)

[4.1.3 Data Augmentation and Noise Injection 18](#_Toc202695136)

[4.1.4 Hold-Out Test Sets 19](#_Toc202695137)

[4.2 Training Dataset Segements Augmentations: 19](#_Toc202695138)

[4.3 Project CNN Architecture for Micro-Doppler Spectrogram Classification ……………………………………………………………………………………..20](#_Toc202695139)

[4.4 Software Description 22](#_Toc202695140)

[4.4.1 Software Description 22](#_Toc202695141)

[4.4.2 Project Folder Structure 22](#_Toc202695142)

[4.4.3 Software Pipeline Flow 23](#_Toc202695143)

[5 Analysis of results 24](#_Toc202695144)

[5.1 CNN Training and Validation Accuracy - Peak Validation Performance 24](#_Toc202695145)

[5.2 CNN Training and Validation Loss - Peak Validation Performance: 25](#_Toc202695146)

[5.3 DopplerTag Confusion Matrix 26](#_Toc202695147)

[5.4 CNN Validation Accuracy Across Datasets with Fixed SNR Levels 27](#_Toc202695148)

[5.5 Results of the Alternative Algorithm for the Project - The Theoretical Model from the Paper‎ [8.1] ([2.5.5 ] Presented in the Theoretical Background Section) 29](#_Toc202695149)

[5.5.1 CNN Validation Accuracy - Peak Validation Performance 29](#_Toc202695150)

[5.5.2 Results of the Alternative Algorithm for the Project - CNN Validation Accuracy Across Datasets with Fixed SNR Levels: 29](#_Toc202695151)

[5.6 Comparison Between the Theoretical Model from the Paper‎ [8.1] and Our Project Model 30](#_Toc202695152)

[6 Conclusions and further work 32](#_Toc202695153)

[1.1 Summary Conclusions: Model Strengths and Practical Contributions 32](#_Toc202695154)

[6.1.1 Model Robustness to Real Noise & Data 32](#_Toc202695155)

[6.1.2 Reliable Learning & Evaluation 32](#_Toc202695156)

[6.1.3 Smart Design & Targeted Optimization 32](#_Toc202695157)

[6.2 Future Work – Proposed Direction 33](#_Toc202695158)

[6.2.1 Transfer Learning with Partial Fine-Tuning: 33](#_Toc202695159)

[6.2.2 Quantitative Assessment of Cross-Set Diversity 33](#_Toc202695160)

[7 Project Documentation 34](#_Toc202695161)

[8 References 35](#_Toc202695162)

[8.1 D. Vovchuk, M. Khobzei, V. Tkach, O. Eliiashiv, O. Tzidki, K. Grotov, A. Glam, and P. Ginzburg, “Micro-Doppler-Coded Drone Identification,” *arXiv:2402.04368*, Feb. 2024. 35](#_Toc202695163)

**List of figures**

[Figure 1: Block diagram 5](#_Toc202634135)

[Figure 2: Illustration of a Radar Sensing Scheme Showing Incident and Scattered Fields from a Target: 8](file:////Users/rotemavivi/Documents/Template%20Project%20Report_FINAL%20(2).docx#_Toc202634136)

[Figure 3: Example of a drone from class 11001010: 8](file:////Users/rotemavivi/Documents/Template%20Project%20Report_FINAL%20(2).docx#_Toc202634137)

[Figure 4: spectrogram examples 8](file:////Users/rotemavivi/Documents/Template%20Project%20Report_FINAL%20(2).docx#_Toc202634138)

[Figure 5: Project Folder Structure 18](#_Toc202634139)

[Figure 6: Examples of Training Dataset Segments Augmentations 19](#_Toc202634140)

[Figure 7: Project CNN Architecture for Micro-Doppler Spectrogram Classification 21](#_Toc202634141)

[Figure 8: CNN Training and Validation Accuracy - Peak Validation Performance 24](#_Toc202634142)

[Figure 9: CNN Training and Validation Loss - Peak Validation Performance 25](#_Toc202634143)

[Figure 10: DopplerTag Confusion Matrix 26](#_Toc202634144)

[Figure 11: CNN Validation Accuracy Across Datasets with Fixed SNR Levels 27](#_Toc202634145)

[Figure 12: Results of the Alternative Algorithm for the Project - CNN Validation Accuracy Across Datasets with Fixed SNR Levels 29](#_Toc202634146)

**List of tables**

[Table 1 : CNN Key Hyperparameters 12](#_Toc202634095)

[Table 2: Selected Hyperparameters for Project Implementation 20](file:////Users/rotemavivi/Documents/Template%20Project%20Report_FINAL%20(2).docx#_Toc202634096)

[Table 3: Comparison Between the Theoretical Model from the Paper‎ and Our Project Model 30](#_Toc202634097)

**List of equations**

[Equation 1- SNR Formula:…………..………………………………………….. 9](#_Toc202694938)

[Equation 2 - ReLU function: 10](#_Toc202694939)

Abstract

**Motivation**

Growing drone use demands detection beyond optical/thermal limits in bad weather, urban clutter, or among other UAVs. A TAU Radar Lab study showed that arranging dipole tags on rotor blades yields unique 8-bit

micro-Doppler codes. This method leverages enhanced Scattering to overcome low RCS and reliably identify individual drones

under all conditions.

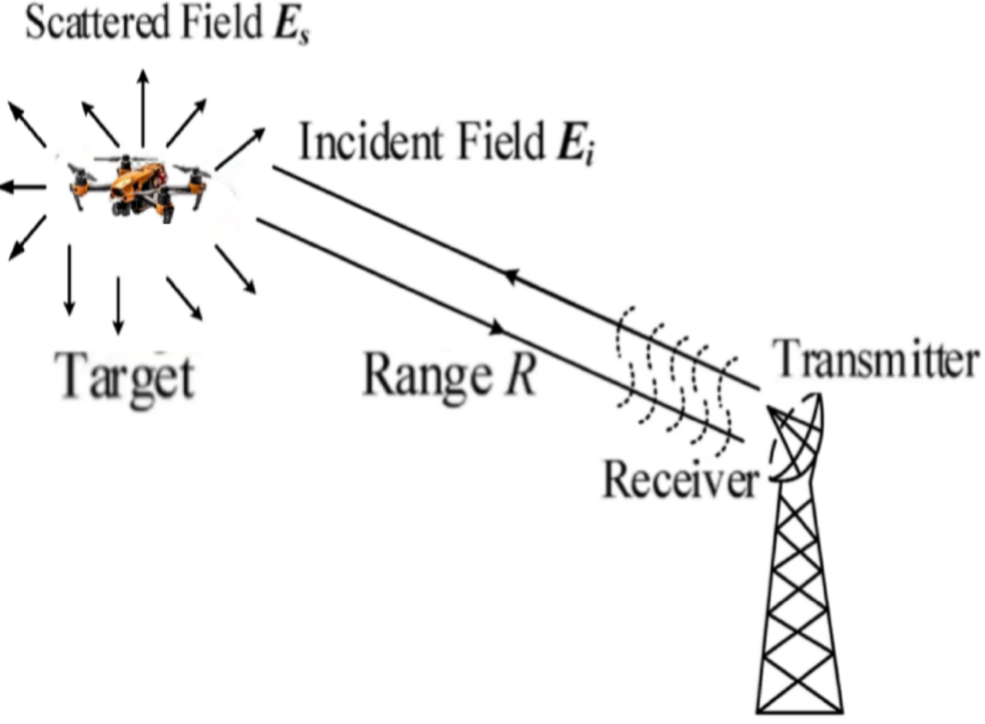
**Project Topic and Objective**

Inour project, we implement the Classifier component of this research, using measured micro-Doppler signatures. Our objective is to accurately classify noisy micro-Doppler signals at low SNRs ratios. To achieve this, we employ a

convolutional neural network (CNN) trained with a synthetic data pipeline

and extensive augmentation, demonstrating the robustness of our approach in noisy SNR environments.

**Project Block diagram**



**Signiture Spectogram**

**Micro-doppler analysis**

**CNN Classifier**

**C/D**

תמונה שמכילה קו, מלבן, צבעוני, עיצוב

תוכן שנוצר על-ידי בינה מלאכותית עשוי להיות שגוי.

**10000000**

תמונה שמכילה צילום מסך, אדום, מלבן, אדום סגול

תוכן שנוצר על-ידי בינה מלאכותית עשוי להיות שגוי.

**10000000**

**Class Number**

Figure 1: Block diagram

# Introduction

## תמונה שמכילה בחוץ, תחבורה, מטוס, שמיים תוכן שנוצר על-ידי בינה מלאכותית עשוי להיות שגוי.Motivation

Today, the use of drones and UAV (unmanned aerial vehicle) systems is steadily increasing in both military and civilian domains, offering effective solutions across a wide range of fields.

As these technologies become more prevalent, there is a growing need to develop reliable control and monitoring systems that can accurately detect and distinguish between UAVs with different purposes—especially under adverse conditions where optical and thermal cameras alone prove insufficient.

### תמונה שמכילה ציפור, נדידת ציפורים תוכן שנוצר על-ידי בינה מלאכותית עשוי להיות שגוי.תמונה שמכילה עיגול, תרשים, עיצוב תוכן שנוצר על-ידי בינה מלאכותית עשוי להיות שגוי.תמונה שמכילה בניין, מטוס, תחבורה, טיסה תוכן שנוצר על-ידי בינה מלאכותית עשוי להיות שגוי.תמונה שמכילה שמיים, מטוס, בחוץ, טיסה תוכן שנוצר על-ידי בינה מלאכותית עשוי להיות שגוי.Challenges in Drone Detection and Classification:

1. **Adverse Weather Conditions**- harsh weather can severely affect drone detection, especially when using optical or thermal methods.

2. **Low-Altitude Flight-** ground reflections and environmental noise complicate detection, requiring resilience to high signal-to-noise ratio (SNR).

3. **Low Radar Cross-Section (RCS)-** drones produce weak and small radar signatures (reflected radar signals or echoes) that make detection difficult.

4. **Distinguishing Between Similar Objects-** radar signatures of drones often closely resemble those of birds or other drones, complicating their identification

### Solution to Challenges:

A study conducted at the Radar Laboratory of Tel Aviv University (TAU) has shown that by affixing dipole tags in specific orientations to a drone’s rotor blades, each configuration yields a distinctive micro-Doppler signature which can be encoded as an 8-bit binary string. This technique both overcomes the low radar cross-section (RCS) of small UAVs-by exploiting enhanced scattering returns - and provides a means of identifying individual drones based on tag orientation, regardless of adverse Weather Conditions, low visibility ,lighting conditions and noisy environments.

## Project Objective: Accurate Classification of Noisy Micro-Doppler Signals Using CNN

In our project, we implement the classifier component of this research, using measured micro-Doppler signatures from 34 unique tag configurations. Our objective is to accurately classify noisy micro-Doppler signals at low signal-to-noise ratios (SNRs). To achieve this, we employ a convolutional neural network (CNN) trained with a synthetic data pipeline and extensive augmentation, demonstrating the robustness of our approach in challenging SNR environments.

# Theoretical background

## Doppler tags-

* Theseelectromagnetic tags modulate the drone’s Doppler radar signal to generate a unique and identifiable 8-bit signature. each signature represents drones of the same type.

Figure 2: Illustration of a Radar Sensing Scheme Showing Incident and Scattered Fields from a Target:

Figure 3: Example of a drone from class 11001010:

תמונה שמכילה קו, תרשים, אנטנה

תוכן שנוצר על-ידי בינה מלאכותית עשוי להיות שגוי.



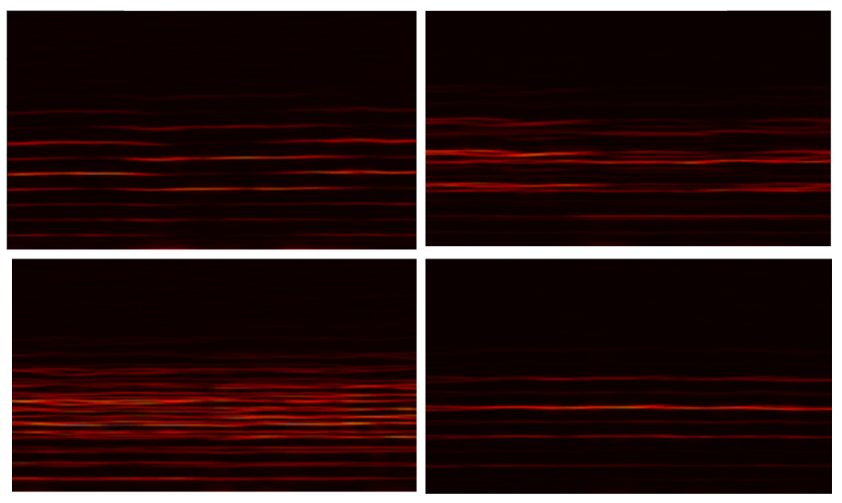
**The dipole tag**

תמונה שמכילה טקסט

תוכן שנוצר על-ידי בינה מלאכותית עשוי להיות שגוי.

## Spectrogram-

* Each sampled signal taken over a duration of 10 seconds, consisting of 50,001 complex values samples ,which their pattern represented by 8-bit signature. The sampled signal is presented as **spectrogram** – 3D matrix of frequency versus time versus magnitude (FFT of the signal).



**11111111**

**10000000**

**00001111**

**10101010**

Figure 4: spectrogram examples

## SNR

* SNR (Signal-to-Noise Ratio) is a measure that compares the level of a desired signal to the level of background noise.
* Mathematical Definition:

Equation 1- SNR Formula:

* SNR Impact
* High SNR: The signal is stronger than the noise 🡪 indicates high quality and improves the reliability of detection and classification.
* Low SNR: Noise is comparable to or stronger than the signal 🡪 difficult to detect, prone to misclassification.

## Synthetic Data

### The Necessity of Synthetic Data for Micro-Doppler Drone Classification:

**Micro-Doppler analysis** is a powerful technique for identifying motion patterns generated specifically by rotating drone blades with attached electromagnetic tags. However, it faces two key challenges: a lack of diverse, labeled data and high sensitivity to noise. Since collecting real-world radar data is complex and limited, especially for tagged drones, **synthetic data** becomes essential to train robust classifiers and improve generalization in noisy or varied environments.

### Creating Synthetic Data: Techniques and Methods:

creating synthetic data increases the size and diversity of training data, improving model generalization and robustness.

Techniques include:

* Adding Gaussian noise to simulate varying SNR conditions.
* Time shifting to represent different flight phases.
* Affine transformations (rotation, scaling, translation) applied to spectrograms.
* Overlapping window segmentation.

### Time Masking technique:

**Time Masking** is a data augmentation technique commonly used in spectrogram-based models. It involves masking (i.e., removing or zeroing out) a continuous time **interval** in the spectrogram. This encourages the model to learn more robust and generalizable features by preventing over-reliance on specific time-localized information. Time masking is particularly useful for simulating real-world scenarios where parts of the signal may be missing or corrupted due to noise or interference.

## CNN Architecture and Hyperparameters

### CNN Architecture

Convolutional Neural Networks (CNNs) are a class of deep learning models designed to process data with grid-like topology, such as images and spectrograms. CNNs are especially effective for detecting local patterns and features due to their use of convolutional filters that scan the input data.

A typical CNN is composed of the following layers:

* Convolutional Layers:
* Apply a set of filters (kernels) to the input data to extract local features. Each filter activates in response to specific types of patterns such as edges, curves, or motion signatures.
* Activation Functions:
* Activation functions introduce non-linearity into the network, allowing it to learn complex patterns beyond simple linear relationships.
* ReLU (Rectified Linear Unit) is the most widely used activation function in CNNs. It outputs zero for negative inputs and the input itself for positive values:

**Equation 2 – ReLU function:** .

It is simple, computationally efficient, and helps avoid the vanishing gradient problem.

* Pooling Layers:
* Pooling layers (e.g., Max Pooling) are used to down sample the feature maps, reducing their spatial dimensions and computational complexity while preserving important information.
* Max Pooling applies a small sliding window (typically 2×2) over the feature map, and at each position, it retains only the highest value within the window - representing the strongest activation in that region.
* Dropout Layers:
* Randomly deactivate neurons during training to prevent overfitting and improve generalization.
* Fully Connected Layers:
* Flatten the output from previous layers and use dense connections to perform final classification.

### CNN Key Hyperparameters

|  |  |
| --- | --- |
| Hyperparameter | Description |
| Number of Filters | Each convolutional layer uses multiple filters to extract features from the input. More filters capture more complex patterns but increase computational cost and risk of overfitting. Typical values: 16, 32, 64, 128, 256. |
| Kernel Size | Determines the size of the filter window scanning the input. Smaller kernels (e.g. 3×3) capture fine details, while larger kernels (e.g. 5×5 or 7×7) detect broader features. Often, several small kernels are stacked to deepen representation. |
| Dropout Rate | Regularization technique that randomly deactivates neurons during training to prevent overfitting. Typical values range between 0.25 and 0.5. |
| Learning Rate | The learning rate controls how quickly the model learns by determining the size of the weight updates at each step. A learning rate that is too high can cause the model to overshoot the optimal point and fail to converge, resulting in unstable or diverging training. A learning rate that is too low leads to very slow learning and can get the model stuck without improving. Choosing an appropriate learning rate is critical: Too high – no convergence Too low – no progress A commonly used starting value is . To improve training stability and performance, it is common to adjust the learning rate dynamically using techniques such as: ReduceLROnPlateau – decreases the learning rate when the model's performance stops improving. |
| Batch Size | Defines how many samples are processed before updating the model’s weights. Smaller batches (16–32) improve generalization but slow training; larger batches (128–256) train faster but use more memory and may generalize less effectively. When increasing batch size, it is recommended to adjust the learning rate proportionally. |
| Epochs | One epoch equals a full pass over the entire training set. Multiple epochs allow the network to refine its weights. Too few epochs 🡪 underfitting (model hasn’t learned enough). Optimal epoch count is found through experimentation or early stopping when validation performance plateaus. |
| Optimizer Type | Determines how the model updates weights based on gradients. The optimizer has a strong impact on convergence speed, training stability, and generalization. Examples of optimizers: SGD – Simple and memory-efficient; good generalization, but needs manual tuning. Adam – Adaptive learning rates per parameter and fast convergence; RMSprop – Effective in noisy or non-stationary problems.  Adam is commonly recommended as a choice for CNNs. |
| Padding | Adds extra pixels (usually zeros) around the input so border convolutions have a full receptive field. Modes: 'valid' – no padding, output shrinks; 'same' – zero‑padding keeps output size equal to input. Padding preserves edge information and controls output spatial size. |
| kernel\_regularizer | Adds a penalty term to the weights during training (e.g. ‘regularizers.l2()’). Common options: L2 regularization (weight decay): λ·∑w² discourages large weights. L1 regularization: λ·∑|w| promotes sparsity. Regularization reduces overfitting; typical λ ranges - . |
| Loss Function | A loss function measures the error between the model's predictions and true values. It guides the training process by quantifying how well the model is learning. |

Table 1 : **CNN Key Hyperparameters**

## Alternative algorithms for project implementation

### Alternative Algorithm: K-Nearest Neighbors (KNN)

K-Nearest Neighbors (KNN) is a simple and intuitive classification algorithm that requires no traditional training phase. When predicting the class of a new sample, KNN computes the distance (typically Euclidean) between that sample and all training samples. It then selects the k closest data points and assigns the most common class among them.

KNN can be highly effective when the dataset is clean and collected under controlled conditions, such as in a lab environment. However, the algorithm is extremely sensitive to noise and irrelevant features. When Gaussian noise or other distortions are added to the input data, KNN’s performance tends to degrade significantly. In such cases, it often misclassifies examples, especially when dealing with high-dimensional or complex input patterns.

### Noise‑Robust Alternatives Considered

Several ensemble methods are known to tolerate noisy data better than KNN. Two prominent examples are Random Forest and XGBoost. These algorithms were not used in this project, as our Convolutional Neural Network (CNN) model outperformed typical classical alternatives and was more suitable for our data type.

* + Random Forest - An ensemble of decision trees built on bootstrap samples of the data. Each tree votes for the final class, which reduces variance and improves robustness to noisy features. Random Forests often excel on structured/tabular datasets with mixed feature types.
  + XGBoost - An optimized gradient‑boosting framework that sequentially adds shallow trees, each correcting the errors of the previous ones. Built‑in regularization and shrinkage make it resistant to overfitting and capable of coping with noisy or imbalanced data.

While these models can handle noisy inputs more gracefully, they may not leverage the spatial and temporal patterns present in spectrogram data as effectively as Convolutional Neural Networks. In our experiments, the CNN achieved superior accuracy and generalization on the Doppler Tag dataset, making it the most suitable choice for the project.

### The CNN Model from the DopplerTag Paper and Our Comparative Approach

CNN Model and Techniques based on the paper [[8](#_[1]_D._Vovchuk,).1] *“DopplerTag: Learning Micro-Doppler for Drone Classification"* which presents research conducted at the Radar Lab, Tel Aviv University.

* Experimental setup and data augmentation
* The indoor setup involved 43 binary-coded tag classes.
* Each tag was recorded three times, with each recording lasting 10 seconds.
* The recordings were segmented into overlapping windows of 1.54 seconds to prepare the data for analysis.
* Additionally, synthetic Gaussian noise was added to the signals to simulate various signal-to-noise ratio (SNR) levels, ranging from 0 to 13 dB.
* CNN Architecture
* The network consists of three convolutional layers with 32, 64, and 128 filters, respectively.
* Each convolutional layer is followed by a ReLU activation function, a 2x2 MaxPooling layer, and a Dropout layer.
* The final part of the network includes a fully connected layer with 128 neurons and output layer.
* Our Approach

In this project, we chose to build upon and enhance the CNN presented in the referenced paper. Our first step was to develop a broader, more tailored, and smarter data augmentation pipeline - adapted to the specific challenges of our dataset and classification task. We then thoroughly evaluated the original CNN architecture and explored various hyper-parameter configurations to achieve optimal performance. Both the augmentation strategy and the hyper-parameter optimization process are detailed in the following sections.

# Simulation

In our project, first, we used raw dataset of sampled signals ,performed Digital processing , created synthetic data by augmentations to the dataset.

Second, we created convolutional neural network (CNN) that classify the micro-Doppler signatures from the datasets.

* We were given six 10-second signals of 50,001 complex values ​​from 34 different departments in advance in an indoor environment, overall 204 samples.
* We created the spectrograms using digital signal processing techniques to emphasize the gain at desired frequencies.
* **creating synthetic data:** sequence of actions:

1. **Creating segmentations-** dividing each signal (50,001 values) to 11 segments, each segment consist 7700 values (each segment overlaps in half with the pervious one), Each segment represent now different sample for the cnn model.
2. Creating synthetic data by **time shifting** to the segments.
3. We divided the data in the way that to **80%** of the data used for **training** and **20%** for **validation**.
4. **For each training sample (segment)** we applied additional augmentations by:
5. **Adding SNR values**— specifically, SNR levels that the model should be able to handle.
6. Using an **affine transformation matrix** (such as angle rotation, scaling and slight modification of coordinate values) in a way that slightly alters the spectrogram while still diversifying the data.
7. Adding **time masking**.
8. **For the validation dataset**, we used the test set and applied smaller augmentations, without time masking and used it to evaluate the model, as this keeps a closer similarity to real-world data patterns.

* It is important to note that the data in the test set was never seen by the model during training.
* By these methods (2-5), we generated four additional synthetic samples from each original sample. Based on 204 original samples, we created 11,220 synthetic samples (204 × 11 × 5).
* **Creating the cnn model,** the model will be trained on the newly generated synthetic data, after obtaining the initial results, the aim is to improve both the cnn model and its accuracy through a process of retraining and experimentation.

During each refinement step, we focus on:

1. diversifying the data to achieve better performance.
2. improving the model structure by tuning the hyperparameters.

# Implementation

A diagram of a project folder structure

AI-generated content may be incorrect.

Figure 5: Project Folder Structure

## ****Synthetic Data Pipeline: Preprocessing, Augmentation, and Dataset Splitting****

### Preprocessing and Data Preparation

* Each micro-Doppler signal was divided into overlapping segments of 7,700 samples, with a 50% overlap between consecutive segments.
* From each segment, we generated seven additional time-shifted versions, resulting in eight total variants per original segment.

### Spectrogram Generation and Dataset Splitting

* For each segment, a normalized spectrogram was computed using the Short-Time Fourier Transform (STFT) with a 256-sample Hann window and 50% frame overlap.
* The resulting spectrograms were stratified and split into:
* 70% for training
* 20% for validation
* 10% for hold-out testing

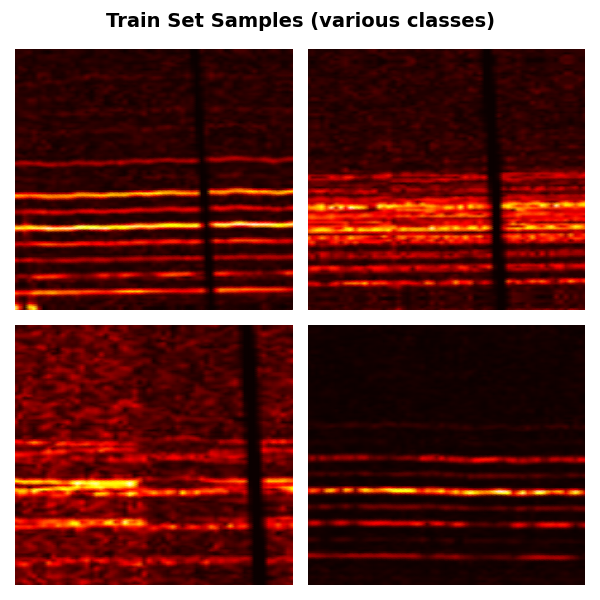
### Data Augmentation and Noise Injection

* To enhance model robustness, the **training set** was augmented with:
* Random affine transformations (±4° rotation, ±10% scaling)
* Time-masking
* Gaussian noise with SNRs randomly sampled from 0 to 32 dB
* For the **validation set**, only mild augmentations were applied:
* Slight affine transformations (±1° rotation, ±2% scaling)
* No time-masking, same SNR range (0–32 dB)

### Hold-Out Test Sets

* From the 10% hold-out set, dedicated evaluation datasets were created at fixed SNR levels.
* These datasets were used to rigorously evaluate model performance under controlled noise conditions.

## Training Dataset Segements Augmentations:



**10000000**

**11111111**

**10101010**

**00001111**

Figure 6: Examples of Training Dataset Segments Augmentations

## Project CNN Architecture for Micro-Doppler Spectrogram Classification

* Following comprehensive hyperparameter optimization and repeated training procedures, we developed a model that maintains a similar architectural structure to the one presented in the paper, while incorporating a refined set of hyperparameters optimized for our specific dataset and classification task. This CNN architecture is common in practical applications because it is both effective and efficient. It is especially well-suited for spectrogram-based classification tasks, such as those involving radar micro-Doppler signals.
* CNN architecture
* The CNN architecture used in this project is a model consisting of three convolutional layers:  
  The first convolutional layer has 32 filters.  
  The second layer has 64 filters.  
  The third layer has 128 filters.  
  Each convolutional layer is followed by a ReLU activation function and a 2×2 max-pooling operation to reduce spatial dimensions. A dropout layer with a rate of 0.3 is also applied to reduce overfitting.  
  After the convolutional blocks, the data is flattened and passed through a dense (fully connected) layer with 128 neurons, followed by a softmax output layer for multi-class classification.

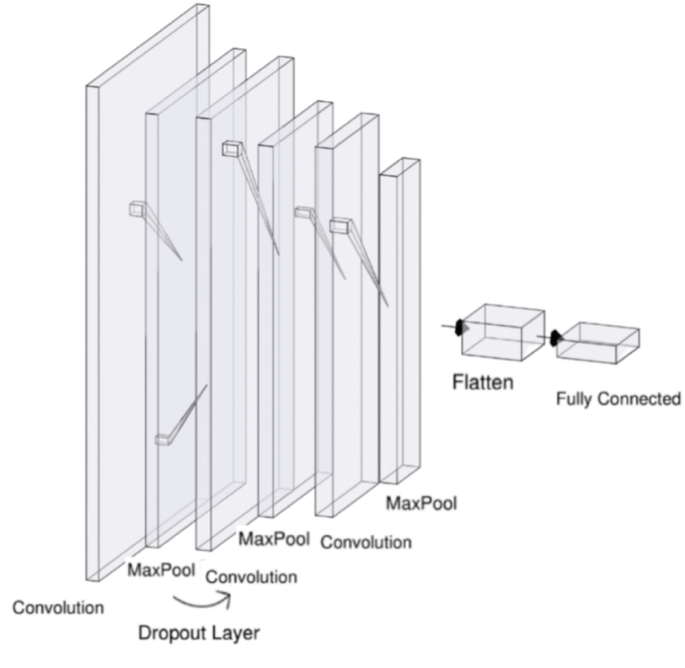


Figure 7: Project CNN Architecture for Micro-Doppler Spectrogram Classification

Selected Hyperparameters for Project Implementation

Table 2: Selected Hyperparameters for Project Implementation

|  |  |  |
| --- | --- | --- |
| Hyper‑parameter | Chosen value | File Reference |
| Number of Filters | 32, 64, 128 | cnn\_3\_layers.py |
| Kernel Size | 3×3 | cnn\_3\_layers.py |
| Dropout Rate | 0.3 | cnn\_3\_layers.py |
| Learning Rate | 10⁻³ | cnn\_3\_layers.py |
| Batch Size | 16 | training.py |
| Epochs | 90 (per run) | training.py |
| Optimizer Type | Adam | cnn\_3\_layers.py& training.py |
| Padding | 'same' | cnn\_3\_layers.py |
| kernel\_regularizer | 10⁻³ | cnn\_3\_layers.py |
| Loss Function | “sparse\_categorical\_crossentropy”  - ideal for multi-class classification with integer labels. | cnn\_3\_layers.py |

## Software Description[[1]](#footnote-1)

### Software Description

* The project was implemented in Python using the Visual Studio Code environment and executed on a Central Processing Unit (CPU). The system development involved signal processing, feature extraction, synthetic data generation, data augmentation, training a Convolutional Neural Network (CNN), and performance evaluation.
* Throughout the development, several widely used libraries were employed:
* NumPy and SciPy – for numerical operations and signal processing, including the use of STFT for extracting spectral features.
* Matplotlib and Seaborn – for visualization of results such as accuracy and loss curves, and confusion matrices.
* OpenCV (cv2) – for image processing and augmentation of spectrograms.
* Scikit-learn – for label encoding, train/test data splitting, and confusion matrix computation.
* TensorFlow / Keras – for defining, training, and evaluating the CNN architecture.
* h5py – for storing and loading datasets in HDF5 format.

### Project Folder Structure

See Figure 5: Project Folder Structure [5]

* The project follows a modular folder structure that separates key stages of the pipeline into distinct components for clarity and maintainability.
* The structure includes:
* data folder – Contains raw data, scripts for generating synthetic datasets (create\_synthetic.py), and tools for spectrogram generation and testing.
* models folder – Includes the CNN architecture implementation (cnn.py).
* training folder – Contains scripts for training (train.py) and evaluating (evaluation.py) the model.
* plot\_utils.py – A utility script for generating accuracy/loss plots and confusion matrices.
* This clear separation supports reusability, debugging, and streamlined development across data preparation, modeling, and evaluation phases.

### Software Pipeline Flow

* The full software pipeline begins with loading raw radar data, followed by the generation of synthetic data to increase diversity and robustness. The resulting dataset is then split into three subsets:
* 70% for training
* 20% for validation
* 10% for hold-out testing
* Following the dataset preparation, the training module is executed to train the CNN model using the training and validation data. Once training is complete, the evaluation module is used to assess model performance on the hold-out test set and to generate relevant metrics and visualizations.
* Remark: Detailed documentation of the software components and accompanying project files, as provided in the project’s README file in ‎project documentation section [7].

# Analysis of results

## CNN Training and Validation Accuracy - Peak Validation Performance

Figure 8: CNN Training and Validation Accuracy - Peak Validation Performance

* After extensive hyperparameter tuning and repeated training runs, our CNN reached its highest validation accuracy at epoch 90, achieving a final validation accuracy of 95.87%, with the training and validation accuracy curves nearly overlapping — indicating strong generalization and minimal overfitting.
* The accuracy curves show a stable and consistent learning process, with rapid convergence in the early stages and minimal fluctuations in the later phases of training. This progression reflects the model’s ability to improve effectively during training and to maintain reliable performance once convergence is reached.

## CNN Training and Validation Loss - Peak Validation Performance:

תמונה שמכילה טקסט, קו, עלילה, תרשים

תוכן שנוצר על-ידי בינה מלאכותית עשוי להיות שגוי.

Figure 9: CNN Training and Validation Loss - Peak Validation Performance

* The loss curves show a rapid decrease followed by stabilization, with close alignment between training and validation losses - demonstrating effective generalization without overfitting. This behavior is consistent and symmetrical with the accuracy curves shown in Figure 8.

## DopplerTag Confusion Matrix

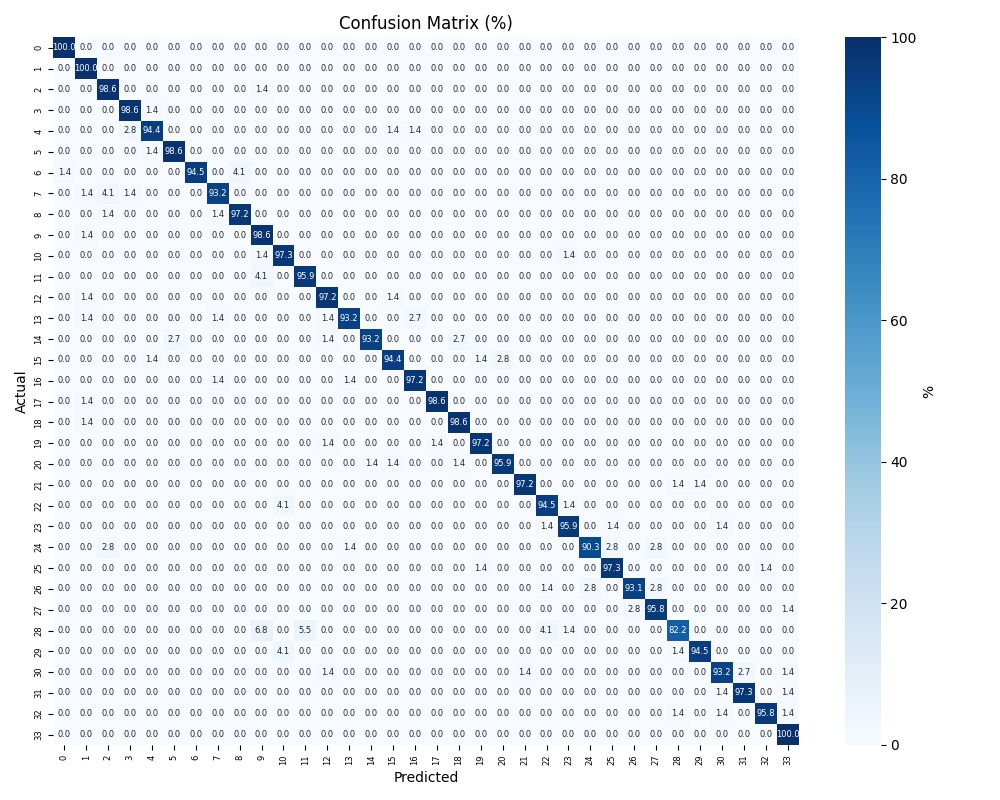


Figure 10: DopplerTag Confusion Matrix

* The **confusion matrix** is a table that visualizes how accurately the model distinguishes between different electromagnetic (EM) tag classes based on micro-Doppler radar signatures.
* What the Confusion Matrix Represents:
* It is a table that compares the true tag class (row) with the predicted class (column).
* Diagonal entries represent correct classifications.
* Off-diagonal entries show where the model predicted the wrong class.
* Confusion Matrix Insights:
* Used for the 34-class indoor dataset.
* Shows an overall accuracy of 95.87% — meaning most predictions were correct.
* Most misclassifications occurred between tag codes with a Hamming distance of 1 (differing by one bit).
* This suggests the CNN struggles mainly when tag codes—and their radar signatures—are very similar.
* Conclusion:  
  The confusion matrix demonstrates the model’s high classification performance while also revealing where and why errors occur. It helps us understand which classes are more likely to be confused and guides improvements in model design.

## תמונה שמכילה טקסט, צילום מסך, קו, תרשים תוכן שנוצר על-ידי בינה מלאכותית עשוי להיות שגוי.CNN Validation Accuracy Across Datasets with Fixed SNR Levels

Figure 11: CNN Validation Accuracy Across Datasets with Fixed SNR Levels

* Summary of the Figure
* The figure demonstrates how Signal-to-Noise Ratio (SNR) influences the classification accuracy of micro-Doppler spectrograms.
* Left Side – Spectrogram Examples:
* Each column represents a different signal class, and each row corresponds to a different SNR level, decreasing from top to bottom.
* As the SNR decreases, the spectrograms become increasingly dominated by noise, making the class-specific patterns less visually discernible.
* Right Side – Hold-out Accuracy vs. SNR:
* The graph presents the validation accuracy of our CNN model on datasets constructed with varying constant SNR levels.
* The curve shows that accuracy improves significantly as SNR increases, with a sharp rise between low and moderate SNR values.
* The accuracy curve demonstrates that our classifier exceeds 90 % once the SNR reaches 15 dB, and climbs to 97 % at 24 dB and above.
* This trend highlights a key advantage of the model: its robust performance under realistic and challenging noise conditions, ensuring reliable classification without the need for exceptionally clean signals.

## Results of the Alternative Algorithm for the Project - The Theoretical Model from the Paper‎ [8.1] ([2.5.5 ] Presented in the Theoretical Background Section)

### CNN Validation Accuracy - Peak Validation Performance

* In indoor experiments across 43 electromagnetic tag classes, the CNN model achieving a final validation accuracy of 97.51%.

### תמונה שמכילה טקסט, צילום מסך, קו, עלילה תוכן שנוצר על-ידי בינה מלאכותית עשוי להיות שגוי. Results of the Alternative Algorithm for the Project - CNN Validation Accuracy Across Datasets with Fixed SNR Levels:

Figure 12: Results of the Alternative Algorithm for the Project - CNN Validation Accuracy Across Datasets with Fixed SNR Levels

CNN– Classification Accuracy Across SNR Levels:

* Left Panel – Spectrogram Samples:

Spectrograms from different binary tag signals are displayed at 10 dB, 15 dB, and 20 dB: At 10 dB, the signals are heavily obscured by noise. At 15 dB, structural patterns begin to appear. At 20 dB, the class-specific features become clearly visible.

* Right Panel – Accuracy vs. SNR:

CNN shows a gradual and consistent improvement in accuracy: starting from ~25% at low SNR, increasing to ~55% at 10 dB, ~75% at 12 dB, exceeding 90% at around 15 dB, and reaching a maximum of 97.51% by 17-18 dB.

The CNN model demonstrates consistent and gradual improvement as signal quality increases.

## Comparison Between the Theoretical Model from the Paper‎ [8.1] and Our Project Model

|  |  |  |  |
| --- | --- | --- | --- |
| **Aspect** | **Baseline Model from the Paper** | **Project Model** | **Advantage** |
| **CNN Architecture** | **main architecture:**  **Conv layers: 32, 64, 128 filters, ReLU activations, 2×2 MaxPooling, Dropout 0.3, Flatten, Fully connected layer (128), Softmax output** | **While the main architecture remained the same, the model incorporates a refined set of hyperparameters, selected through thorough tuning and tailored to the characteristics of our dataset and classification goals.** | ✓ Optimized Design |
| **Accuracy vs. SNR** | 25% at low SNR ,  ~90% at 15 dB ,  reaching 97.5% at 17–18 dB and above | ~55% at 10 dB,  ~75% at 12 dB,  ~90% at 15 dB,  reaching 97% at 24 dB and above | – |
| **SNR Range** | SNR range from 0 to 13 dB | SNR range from 0 to 32 dB | ✓ Broader noise tolerance |
| **Data Augmentation** | Time shifting, Gaussian noise | Time-masking, affine transformations (scaling, rotation, translation), broader noise range | ✓ Greater variability and signal diversity |
| **Synthetic Data & Diversity** | Not used | Structured and realistic synthetic data generation exposing the model to challenging conditions | ✓ Enhanced generalization and robustness |
| **Overfitting Prevention** | Not applied | Validated by tight overlap between training and validation accuracy curves | ✓ Demonstrated learning stability |
| **Evaluation & Data Split Strategy** | No structured matching of synthetic data to specific dataset splits | Synthetic data was carefully matched and stratified across training, validation, and test sets, with tailored augmentations per split | ✓ Improved consistency and fair evaluation |
| **Exposure of Validation/Test Data** | Not discussed | Validation and test data were not seen during training | ✓ **Ensures unbiased evaluation by preventing data leakage between training, validation, and test sets.** |

Table 3: Comparison Between the Theoretical Model from the Paper‎ and Our Project Model

While the CNN model presented in the paper may achieve similar accuracy to the one developed in this project, this does not indicate equivalent performance, as detailed in the table above. The enhancements introduced in our approach—including structured and diverse synthetic data generation, robust overfitting prevention techniques, and a carefully designed data splitting strategy—contribute to improved generalization, stability, and evaluation reliability. These factors provide a more rigorous and trustworthy learning pipeline compared to the methodology described in the paper.

**Remark:** A more detailed comparison between the theoretical model and our project model, along with additional insights, is provided in the later conclusions section in [‎6.1].

# Conclusions and further work

## Summary Conclusions: Model Strengths and Practical Contributions

### Model Robustness to Real Noise & Data

* Demonstrates strong model robustness across a wide range of SNR levels, including challenging noise conditions.
* Advanced augmentation methods, such as time-masking, affine transformations, and variable SNR noise, were applied to further enhance robustness to signal variability.
* The synthetic data was designed to be not only realistic, but also more diverse and noisier than real-world data, exposing the model to harsher conditions and improving its resilience to extreme scenarios.

### Reliable Learning & Evaluation

* Close alignment between training and validation accuracy curves demonstrates consistent learning and minimal overfitting.
* Evaluation was performed using fixed-SNR datasets, allowing for fine-grained analysis of model robustness.
* Validation relied on a truly unseen dataset, ensuring clean separation from training data and preventing data leakage.

### Smart Design & Targeted Optimization

* The CNN architecture was selected through extensive hyperparameter tuning.
* Data was split into training, validation, and hold-out test sets with customized augmentation strategies per subset:
  + - Training data underwent broader, more aggressive augmentations to improve model resilience.
    - Validation and test sets received lighter augmentations to preserve resemblance to real-world signal conditions.

## Future Work – Proposed Direction

### Transfer Learning with Partial Fine-Tuning:

The current model demonstrates **strong performance and robustness** across a wide SNR range, including low-SNR conditions. Integrating transfer learning with partial fine-tuning into the existing model could **further improve adaptability and efficiency**, especially in scenarios with even more degraded signal quality.  
**This approach** uses pretrained models while adapting only selected layers to the task, enabling **efficient training** and maintaining **strong generalization**. It may **further improve classification accuracy — reaching 99–100% —** and enhance the model’s ability to operate reliably under **even lower SNR conditions**.

### Quantitative Assessment of Cross-Set Diversity

The current model already ensures proper separation between training, validation, and test sets — with tailored data augmentations and unseen evaluation data.  
As a future direction, one could explore the use of **quantitative tools to further validate and analyze the diversity** between these datasets.  
Measuring differences in signal distributions — such as spectral content, temporal dynamics, and noise characteristics — can **complement the current evaluation process** and offer a deeper understanding of how the model generalizes across varied and realistic conditions.

# Project Documentation

* The Documentation location link: (<https://github.com/amitstein9/Drone-Identification-Using-Micro-Doppler.git>)
* Section [4.4 Software Description] provides a description of the software documentation and associated project files.
* Detailed documentation of the software components and accompanying project files, as provided in the project’s README file at the project link.

# References

## Paper: D. Vovchuk, M. Khobzei, V. Tkach, O. Eliiashiv, O. Tzidki, K. Grotov, A. Glam, and P. Ginzburg, “Micro-Doppler-Coded Drone Identification,” *arXiv:2402.04368*, Feb. 2024.

1. [↑](#footnote-ref-1)