UNIVERSITY OF OKLAHOMA

GRADUATE COLLEGE

DEEP LEARNING FOR WEAK TARGET DETECTION IN RANGE-DOPPLER DATA

A THESIS

SUBMITTED TO THE GRADUATE FACULTY

in partial fulfillment of the requirements for the

Degree of

MASTER OF SCIENCE

BY

Bibi M Dang
Norman, Oklahoma
2022

DEEP LEARNING FOR WEAK TARGET DETECTION IN RANGE-DOPPLER DATA

A THESIS APPROVED FOR THE
SCHOOL OF COMPUTER SCIENCE

BY THE COMMITTEE CONSISTING OF

Acknowledgements

I would like to thank my co-advisors Dr. Justin Metcalf and Dr. Dean Hougen for their continuous support and for the opportunity to collaborate on an interesting subject. I would also like to thank Dr. Dimitrios Diochnos for his valuable feedback as a member of my committee. Thank you to the Army Research Laboratories for sponsoring this work under Subcontract to Huntington Ingalls Industries. However, the views expressed in this thesis are those of the author and do not reflect the official policy or position of ARL, or the U.S. Government. No official endorsement by ARL should be inferred. Lastly, I would like to thank the friends and family whose support made this possible.

Abstract

A consistent issue for detectors in radar systems is how to correctly distinguish target signals from random noise. This is especially true for weak targets with low signal-to-noise ratios (SNRs). Traditional target detection techniques, such as constant false alarm rate (CFAR) detectors, apply detection thresholds that must be set to maximize the probability of detection (P_D) while minimizing the probability of false alarm (P_{FA}). These traditional detection techniques also struggle with increasing levels of computational complexity in low-SNR environments. This work investigates the application of deep neural networks towards the radar target detection problem. Two neural network architectures, NoisyLSTM and U-Net, are tested on range-Doppler data to identify regions of interest in which targets may be present. The U-Net model demonstrates promising results, producing detection predictions with a P_D of 0.97 and P_{FA} of 0.01 for targets captured by a staring radar at 10dB input SNR. This deep learning architecture may serve as a valuable preprocessing step to reduce the search space of more sophisticated radar detectors.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

A fundamental radar task is to correctly distinguish target signals from random noise. This is a relatively simple problem in cases where target signals stand out clearly among noise. However, target detection quickly becomes difficult for weaker targets. Traditional detectors in radar signal processors attempt to determine if a specific signal returned by a radar corresponds to a real target. These detectors often utilize a threshold detection technique in which signals higher than a predefined threshold are classified as targets and signals lower than the threshold are assumed to be noise. This can further be considered as a binary hypothesis test where H_0, the null hypothesis, assumes that no target is present among returned noise, and H_1 assumes a target is present in combination with noise (Gusland et al., 2020). For a given signal value X and threshold value T, the null hypothesis is accepted when $X \leq T$ and H_1 is accepted when $X > T$.

However, determining the threshold for these detection techniques can become challenging. Setting a threshold too low will often result in signals from random noise spikes being misclassified as targets. These misclassified signals are referred to as false alarms. Conversely, setting a threshold too high will exclude most noise and interference signals, but it may also fail to identify true target signals entirely (Gusland et al., 2020). This problem becomes even more complicated for weak targets with low signal-to-noise ratio (SNR). Detection algorithms are constantly seeking a balance to maximize the probability of detection, P_D, while minimizing the probability of false alarms, P_{FA}. In many cases, a constant-false-alarm rate (CFAR) detector is used to dynamically set and adjust the detection threshold to maintain a constant P_{FA}. In these cases, thresholds are set to maximize the P_D at a given SNR. However, traditional CFAR techniques cannot readily detect low-SNR targets.

Several detection techniques have been proposed to attack the weak target detection problem, often by attempting to improve the output target SNR. One example of such a technique is track-before-detect. Traditionally, the threshold value described above is applied to every measurement frame returned by the radar. However, track-before-detect processes data across several frames before deciding whether or not a target is present (Grossi et al., 2013). This technique accomplishes two tasks. First, the coherent target signal is amplified by combining the consistent, albeit weak, signal returned by the target across several measurements while mitigating the effects of fluctuating noncoherent noise signals. Second, a "track" is returned at the same time as detection. This occurs as the combined radar returns showcase the detected target's movement over time. In the case of low-SNR targets, a target may only appear as a point in space and barely distinguishable from surrounding noise. However, by examining multiple return frames, the range or Doppler walk of the target over time introduces a unique "spatial" element to target signatures (Gusland et al., 2020). This can make targets easier to detect, but the number of possible target trajectories to explore explodes exponentially as the number of combined frames increases.

Previous work to improve target SNR, such as track-before-detect, has inspired the work outlined in this thesis. Recent advancements in the field of computer vision have proven that deep neural networks are an excellent tool for identifying spatial features in image and video data. These networks may also prove useful in target detection by exploiting the temporal nature of radar data to isolate unique features of targets amongst noise.

1.1 Research Objectives

The primary goal of this research is to reduce the computational complexity of track-before-detect-algorithms by incorporating deep learning models as a preprocessing step. These models can serve as a quick and efficient method to predict likely regions of interest for targets and reduce the search space of complicated signal processing techniques. This thesis outlines and analyzes two neural network architectures, NoisyLSTM (Wang et al., 2021) and U-NET (Ronneberger et al., 2015), that can be modified and applied to radar data for target detection. The primary objectives of this thesis are as follows:

- Generate and prepare simulated radar data to be used in two deep learning models: NoisyLSTM and U-Net
- Implement and modify NoisyLSTM and U-Net models to apply to the target detection task
- Train viable models using representative training and validation datasets across varying SNR values
- Test viable models on unseen data and evaluate performance based on P_D and P_{FA} metrics

- Assess feasibility of models as a preprocessing step across varying SNR values

1.2 Contributions to the Research Community

This thesis analyzes the application of deep learning towards the radar target detection problem. More specifically, two neural network architectures, NoisyLSTM and U-Net, were adapted and applied to range-Doppler radar data. Both architectures, originally designed as segmentation models for computer vision tasks, were capable of processing range-Doppler data similarly to images or video frames in an attempt to identify unique target features over time. Results produced by the U-Net model in particular suggest that deep learning approaches to target detection can be used to improve current signal processing approaches as a preprocessing step used to highlight regions in which a target may be present. This form of preprocessing can then reduce the search space and computational complexity of traditional detection techniques.

1.3 Thesis Organization

This thesis is organized into 6 chapters. Chapter 1 outlines the target detection problem for radar. Chapter 2 introduces relevant background information on deep learning and neural networks. Chapter 3 details the experimental design used to test both proposed deep learning methods. Chapter 4 presents the results obtained from training and testing. Chapter 5 includes concluding remarks on the overall work. Lastly, Chapter 6 details potential future work and areas for improvement.

Chapter 2

Background

The application of machine learning to radar tasks is becoming an increasingly popular area of research. In most cases, this research attempts to apply machine learning techniques that have proven to be successful in other domains and apply them to radar data. Deep learning and its applications have also shown to be an especially popular focus within this area of research. For example, range-Doppler data is a form of radar data that has shown promising results in classification and detection tasks when processed by deep learning architectures commonly used in computer vision (Wang et al., 2022) (Gusland et al., 2020) (Altmann et al., 2019). This chapter outlines a brief introduction into deep learning concepts, introduces two deep learning architectures applied in this work, and discusses related work.

2.1 Deep Learning and Neural Networks

Deep learning is an area of machine learning based heavily on artificial neural networks. Simply put, these networks are typically composed of several functions that are applied to some form of input data (Goodfellow et al., 2016). For example, say a model is composed of three functions, $f^{(1)}$, $f^{(2)}$, and $f^{(3)}$, chained together so that

$$f(x) = f^{(3)}(f^{(2)}(f^{(1)}(x)))$$

The variable x refers to the original input data, each function $f^{(n)}$ is the n^{th} layer of the network, and $f(x)$ is the output layer. Neural networks are then trained to produce an output $f(x)$ that most closely matches a predefined correct output $f^*(x)$. The functional layers in between the input and output layers are referred to as hidden layers. The learning algorithm must choose how to apply the hidden layers to produce the desired result, but their expected output and behavior is not explicitly defined in the training data (Goodfellow et al., 2016).

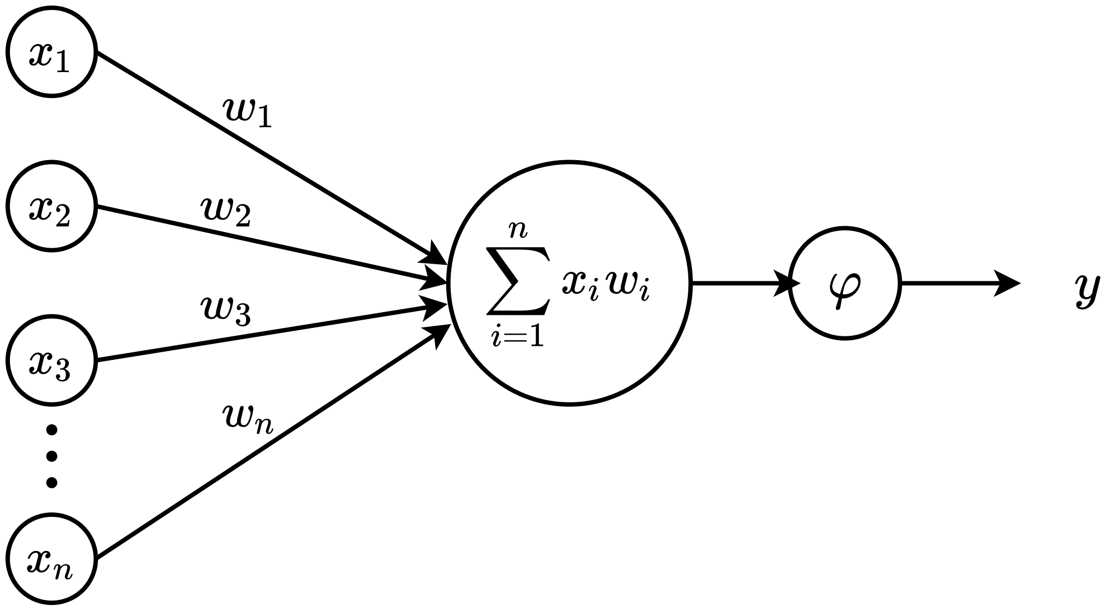

Figure 2.1: Artificial neural network neuron

Neural networks are also commonly organized into a series of processing units called neurons (Kelleher, 2019). At a fundamental level, a neuron maps several inputs to a single output, as seen in Figure 2.1. Within a neuron, the weighted sum of inputs $[x_1, x_2, \ldots, x_n]$ is passed to an activation function φ to produce an output y. There are several popular activation functions used in neural networks today, sigmoid and hyperbolic tangent functions for example, but the primary goal of this step is to add non-linearity to the network in order to model complex relationships (Kelleher, 2019). Figure 2.2 shows a simple fully-connected neural network, meaning the output of all neurons in a given layer are fed as input into every neuron in the following layer.

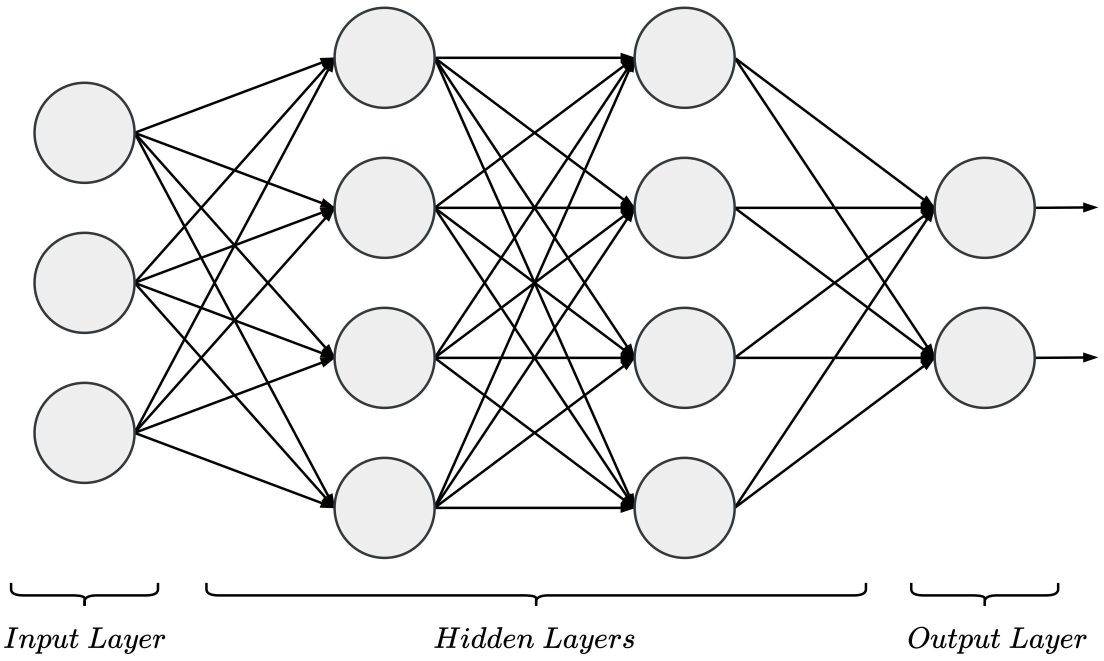

Figure 2.2: Example fully-connected neural network architecture

2.2 Convolutional Neural Networks

One of the most popular deep learning networks used today are convolutional neural networks (CNNs). CNNs perform best on data structured in a grid or matrix as they were originally designed for image recognition tasks (LeCun et al., 1989). A CNN is designed to extract local features of the input data in early layers and combine those features to identify higher-level features in later layers

(Kelleher, 2019). Face recognition in images is a common example used to explain this process. Early layers in a CNN are tasked with identifying simple features, such as edges or curves, that are localized within a small subsection of pixels. The following layers combine these low-level features into more identifiable body parts, such as eyes and ears, until a whole face can be identified in the image (Albawi et al., 2017).

The basic building blocks of CNNs are convolutional layers. Convolutional layers typically consist of three parts: a convolution stage, a nonlinear activation function, and a pooling stage. The convolution stage is based on a mathematical convolution operation, often denoted with an asterisk (*), as shown in Figure 2.3 (Goodfellow et al., 2016). During convolution, a weight matrix, or kernel, is passed over a given input to reduce it into a smaller output referred to as a feature map. This map is then fed into a nonlinear activation function before moving on to the pooling stage.

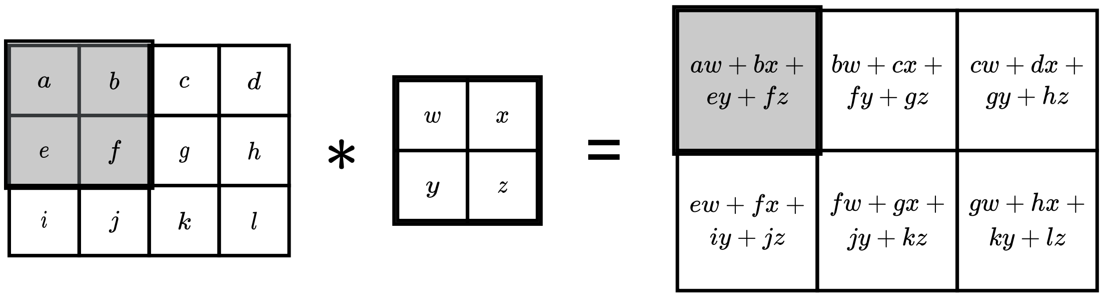

Figure 2.3: Convolution matrix operation

Pooling is an important step to reduce the complexity of a feature map for future layers and to minimize the effect of small variations among samples belonging to the same class (Albawi et al., 2017). The most common pooling function is called max-pooling. Figure 2.4 shows an example of max pooling in which an input is reduced by taking the max value within a given subregion, say 2x2, starting in top-left corner and moving by a stride of 2 across the entire image. By design, these three stage compose a single convolutional layer with the intent of reducing a given input into distinguishable features.

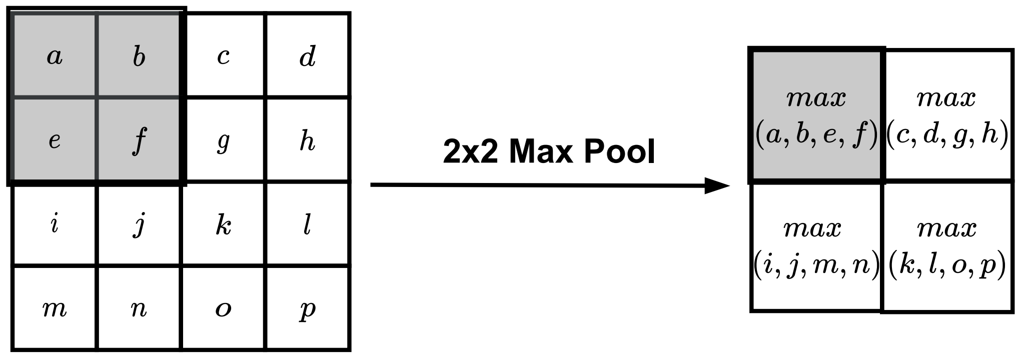

Figure 2.4: Max-pooling operation

2.3 Long Short Term Memory

Another common family of neural networks are recurrent neural networks (RNNs). RNNs are designed to analyze sequential or temporal data using a single hidden layer (Kelleher, 2019). Unlike CNNs, which typically process input data samples individually, RNNs expect a sequence of inputs $[x_1, \ldots, x_n]$ that are unrolled and processed one at a time through a recurrent loop. In many cases, these input sequences are composed of data samples across multiple timesteps. In addition to the sequential input data, RNNs also store all outputs from the hidden layer in a memory buffer to be used later. These hidden layer outputs are referred to as hidden states. RNNs are recurrent in the fact that each input is processed using the hidden state value calculated at the previous timestep. Figure 2.5 demonstrates how the output hidden state, h_t, for each input sample, x_t, is repeatedly fed back into the single hidden layer at each timestep t.

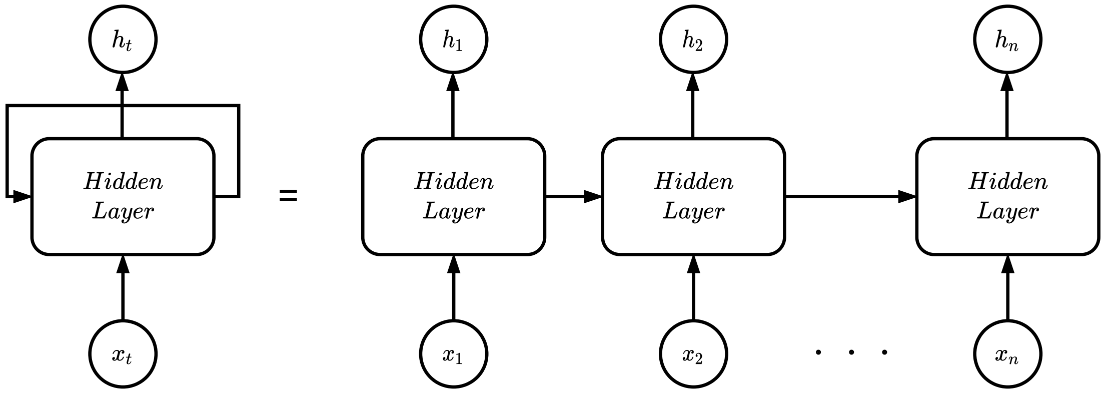

Figure 2.5: Example RNN architecture

A popular example of RNNs are long short-term memory (LSTM) networks (Hochreiter and Schmidhuber, 1997). Early RNNs struggled to scale over long time sequences or to generalize well to new data formats. However, LSTMs have proven to be effective at capturing both long-term and short-term dependencies across several application domains (Greff et al., 2015). LSTMs are made up of a single hidden layer called the LSTM cell. Similar to a simplified RNN, LSTM cells process a series of sequential inputs and hidden states, but it also keeps track of a series of cell states that retain information across all time steps (Goodfellow et al., 2016). Figure 2.6 shows how data moves forward through a single LSTM cell. Each input instance and corresponding hidden state pass through a series of gates: the forget gate, the input gate, and the output gate (Kelleher, 2019). Each gate utilizes a set of biases b, input weights U, recurrent weights W, and a sigmoid function to generate an output (Goodfellow et al., 2016). The outputs of the forget gate f_t, input gate i_t, and output gate o_t at a given timestep are defined as

$$f_t = \sigma(U_f h_{t-1} + W_f x_t + b_f)$$

$$i_t = \sigma(U_i h_{t-1} + W_i x_t + b_i)$$

$$o_t = \sigma(U_o h_{t-1} + W_o x_t + b_o)$$

In addition to the three gates, data may undergo various pointwise multiplication, pointwise addition, or hyperbolic tangent functions before a cell produces an output hidden state and cell state to be used in the next timestep. These additional operations are also shown in Figure 2.6.

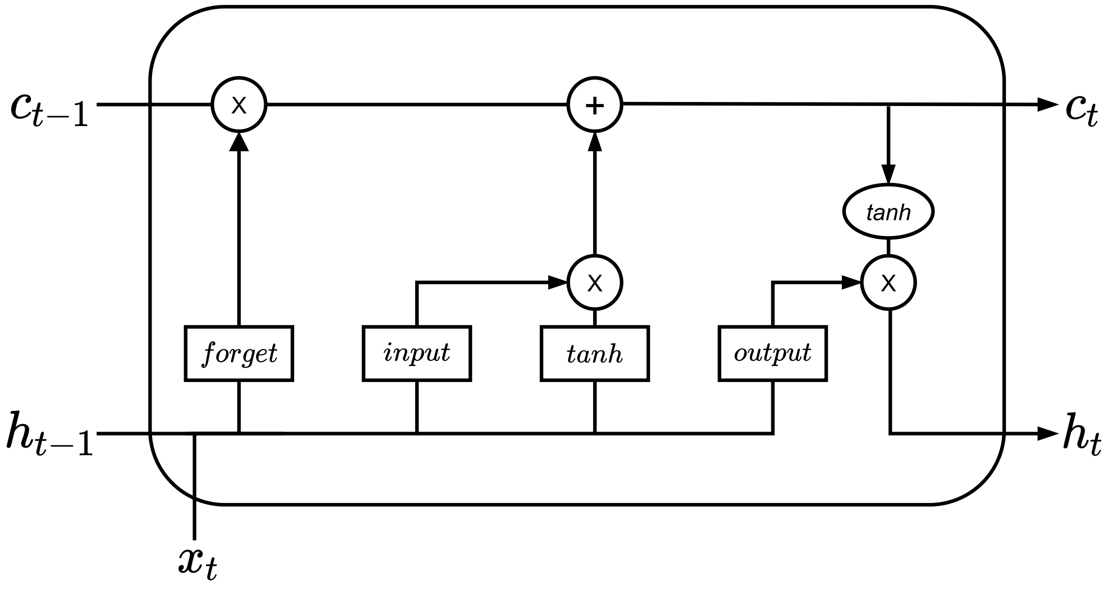

Figure 2.6: LSTM Cell

2.4 Related Work

While traditional signal processing techniques are still the predominant approach in detection literature today, there has been a recent increase in interest for using machine learning techniques for several radar tasks - including target detection. More specifically, deep learning methods are becoming a popular research topic aiming to reduce the complexity of traditional radar tasks, and initial investigations into deep learning for radar data have shown promising results.

CNNs are a popular deep learning architecture that have recently been applied to several target detection applications where target signals may be particularly weak (Wang et al., 2022) (Gusland et al., 2020). These applications are often inspired by advancements in the field of computer vision, where CNNs are a common choice for image classification. In image classification tasks, some deep learning model attempts to classify objects in an image into one or more categories. Target detection in radar can also be considered as a binary classification problem in which a target is either present or not present. Furthermore, existing computer vision architectures can easily be extended to radar data depending on its initial format. For example, YOLO (Redmon et al., 2016) is a very popular CNN-based network widely used in object detection and classification tasks, but the radar community has since extended it to classify signals in spectrograms (O'Shea et al., 2017) and synthetic aperture radar (SAR) images (Cui et al., 2021).

Range-Doppler maps (RDMs) are another form of radar data that can often easily be applied to deep learning models commonly used in computer vision (Wang et al., 2022) (Gusland et al., 2020) (Altmann et al., 2019). Computer vision models traditionally expect an input image of size *(C, H, W)* where C is the number of color channels – $C = 1$ for grayscale images and $C = 3$ for colored images – and *H & W* are the height and width of the image. The input image is further composed of a series of pixel values where some integer ranging from 0 to 255 indicates how dark each pixel appears. In this case, a pixel value of 0 represents the darkest shade of black available, and a pixel value of 255 represents the lightest shade of white available.

An RDM is a 2-D matrix containing signal data relative to distance and velocity. This data structure is very similar to a traditional grayscale image, except that each value in the 2-D matrix represents a radar signal value rather than a pixel color value. Because of this similarity, RDMs can often be interpreted as one-channel grayscale images in existing computer vision models. Analyzing multiple RDMs at a time also incorporates a temporal element to a target's signature. Gusland et al. (2020) introduced an approach to stack multiple RDMs to create a 3-D range-Doppler image that can be fed into a CNN classifier. This approach condenses temporal data into a single image, but other approaches have attempted to utilize video processing techniques to capture temporal features over a longer period of time (Altmann et al., 2019) (Baird et al., 2020). These approaches utilize LSTM models to analyze multiple sequential RDMs at a time similar to analyzing several frames in a video.

Chapter 3

Experimental Design

To test the application of deep learning for target detection, two neural network architectures, NoisyLSTM and U-Net, were applied to simulated range-Doppler data. This chapter outlines how the range-Doppler data was generated and labeled in order to train and test the neural networks. Then, a detailed overview of both the NoisyLSTM and U-Net architectures is provided before discussing which metrics will be used to evaluate each model's performance for this task.

3.1 Simulation Model

Training and testing data for the neural networks was generated using a simple pulse-Doppler radar simulation in MATLAB. Within the simulation environment, all targets were restricted to move directly towards or away from the radar at randomized locations, velocities, and accelerations. Random Gaussian noise was also added to every simulation scenario. Each simulation run produces a set of RDMs as output. A single RDM represents the radar return at a given timestep, and the returned set of RDMs at the end of each run represents a sequential history of radar data over the total observation time. Each RDM also undergoes an additional preprocessing step where all complex values are transformed to real values by taking the magnitude squared of the overall RDM data. It is important to note that this simulation focuses solely on point targets among random thermal noise, and it does exclude several real-world complications such as the effects of pulse compression and Doppler sidelobes. Although this simulation is relatively simple from a radar perspective, it is a flexible framework that allows for quick generation of large amounts of data while also showcasing the learnable features targets exhibit in real radar data.

An example RDM generated by the MATLAB simulation tool is shown in Figure 3.1. Each individual RDM is structured so that Doppler is measured along the x-axis and range is measured along the y-axis. Doppler values, which correspond to relative velocity, at the highest and lowest ends of the x-axis correspond to the highest speeds going towards or away from the radar. Doppler values at the center of the x-axis correspond to no motion relative to the radar. The range values along the y-axis correspond to distance so that increasing values are a farther radial range away from the radar. Figure 3.1 also includes a highlighted target cut-out to demonstrate an example target signature within the RDM. The target appears as a distinctive horizontal streak that occupies a very small area within the RDM as a whole. This streaking indicates the "horizontal walk" of the target that has been captured by the radar across several Doppler bins. In other words, this is an accelerating target, and therefore the Doppler measurement (which measures velocity) changes across the collection of RDMs.

The simulation environment is capable of producing two classes of data, each representing RDMs collected by two simple types of radars. The first class of data is referred to as the *staring* class. This class mimics a stationary radar staring

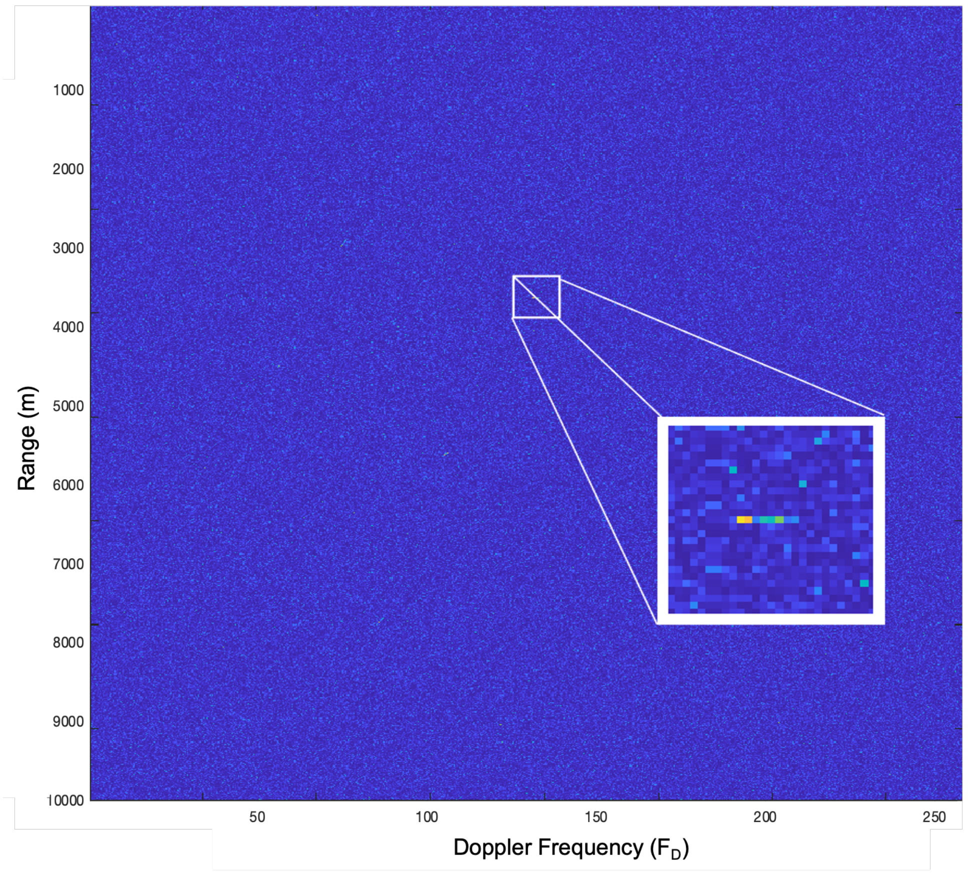

Figure 3.1: Example RDM with target cut-out

at a particular angle, and each RDM is recorded sequentially with no time delay, $T_\Delta = 0$, between collects. The second class is referred to as the *scanning* class. This class represents targets identified by a radar scanning across a given area, and each RDM is collected after a short time delay, $T_\Delta = 75ms$, to compensate for the time it takes the radar to sweep across the given area. In order to add more variety to the training and testing data, targets in each class also differ in acceleration. Targets in the staring class are assigned constant accelerations equal to 7% of the target's initial velocity, and targets in the scanning class are assigned a constant velocity with no acceleration.

Figures 3.2 and 3.3 show examples of targets included in the staring and scanning radar classes at varying SNR levels. While target signatures within each class may vary, the example targets shown in these figures demonstrate some of the most distinguishable cases in order to highlight the main differences among varying SNR levels in each class. The RDMs in these figures are shown using a scaled colormap that maps the range of raw range-Doppler values to a full color spectrum. Target signatures, especially in cases where targets have a high SNR value, will traditionally correlate with higher signal values. This can be seen in the larger or "brighter" pixel values shown in the target cutouts. At all three SNR levels shown, the targets at 30dB SNR appear brightest and are easily distinguishable among the noise in both the staring and scanning radar classes. The 30dB SNR and 10dB SNR examples for the staring radar class are also more discernible due to their horizontal streaking from acceleration. The 3dB SNR examples present a much more challenging problem. At this SNR, targets are much less distinguishable among noise and often lack any discernible streaking shape. In this case, the neural networks will need to rely on some form of amplification gained by examining several RDMs in sequence in order to differentiate fluctuating noise spikes from target signals that will remain relatively consistent over time.

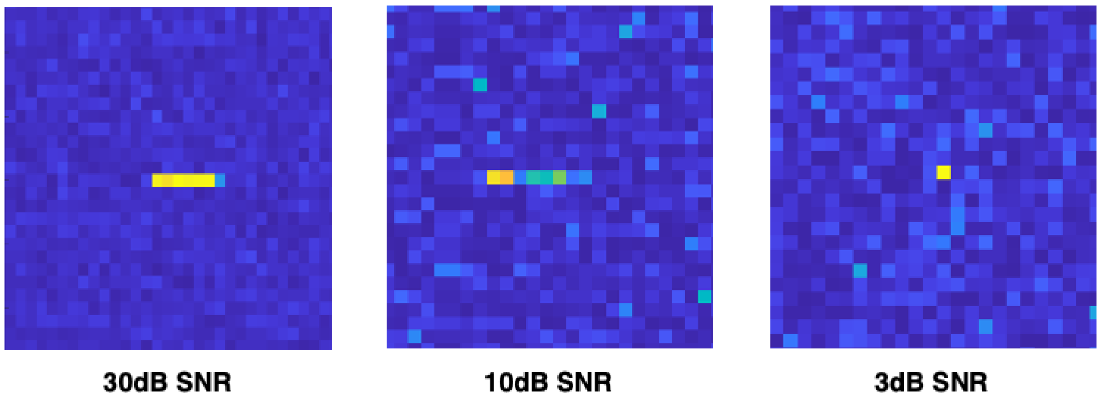

Figure 3.2: Example target signatures for staring radar class at varying SNRs

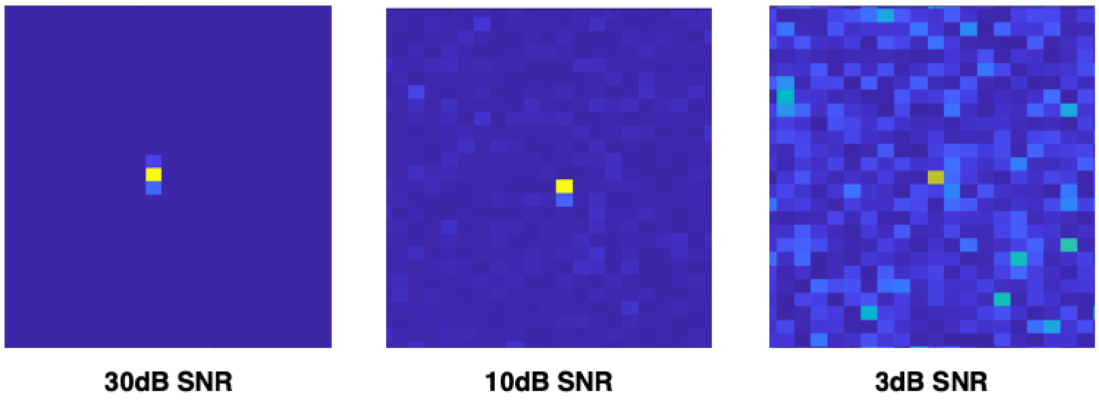

Figure 3.3: Example target signatures for scanning radar class at varying SNRs

3.2 Data Format and Datasets

3.2.1 Generated Simulation Data

Three datasets were created for both the staring and scanning radar classes to simulate target returns at 30dB, 10dB, and 3dB SNR. Targets at 30dB SNR should be easily distinguishable, and they will become increasingly difficult to detect at lower SNR values. Each dataset was generated using the constant radar parameters shown in Table 3.1. The center frequency, F_C, and bandwidth, BW, are important parameters used to govern the range of frequencies used by the radar receiver. The pulse repetition frequency, PRF, and total number of pulses determines how often the radar emits a pulse of energy and how many pulses are used to make a measurement. Lastly, T_Δ refers to the time delay between measurement collections, and T_{Obs} is the total observation time spent collecting measurements. Additionally, each simulation run produced RDMs of size 10,000×256 each containing 100 targets. RDMs of this size would be overwhelming for track-before-detect techniques, but serve as an ideal example of cases in which preprocessing using neural networks may prove to be beneficial.

Table 3.1: Constant radar simulation parameters

	F_C	**PRF**	**Pulses**	**BW**	T_Δ	T_{Obs}
Param.	5 GHz	5 kHz	256	50 MHz	[0, 75] ms	4 s

3.2.2 Sliding Window Approach

After generating the initial datasets using the MATLAB simulation environment, an additional preprocessing step was performed using a sliding window operation on each RDM before being passed over to the neural networks. This preprocessing step standardizes the input size of data being fed into the models and compensates for the possibility of variable RDM sizes. This technique adds an additional benefit of creating square inputs for the neural networks. To accomplish this, each RDM is divided into several smaller non-overlapping windows. For the purposes of this experiment, a 16x16 sliding window was used for each RDM across all generated datasets. The resulting 16x16 RDM windows served as input for the neural networks.

3.2.3 Labels and Segmentation Masks

Neural networks for computer vision can be designed for a wide variety of goal tasks. Traditionally, most computer vision tasks can be organized into three broad categories: classification, localization, and segmentation. In classification, the primary goal of the neural network is to analyze an image, or subregion of an image, and classify it into one or more categories. In the context of this target detection problem, this would be equivalent to a network analyzing a single RDM window and predicting whether or not a target is present anywhere in the window. This approach results in a few problems. First, this form of binary output provides very little information about the scene as a whole. No additional information is provided on where the target is located within a window or how many targets are present within a window. Furthermore, a binary label assigned to each RDM window provides very little information to the neural network during training. While the model will be penalized for target predictions in no-target windows, it will not be penalized if it produces a "correct" window prediction by flawed logic (i.e. if its choice was influenced by a noise spike rather than a target).

Localization and segmentation are similar tasks that often produce more detailed outputs than simple classifiers. In localization, networks are trained to predict bounding boxes around objects in images. In these tasks, a set bounding box coordinates are provided as ground truth labels for each object to be detected during training. Segmentation attempts to perform a similar task, except at the pixel level. Therefore, instead of producing a bounding box prediction, segmentation models produce segmentation masks that label each individual pixel in an image. These masks can also easily be extended to artificially define bounding boxes as a post-processing step. In the context of this target detection problem, this is equivalent to a network producing a 16x16 prediction mask where every cell is either labeled as having no target (0) or a target (1). This will not only provide information on where the target is located in the window, it will also provide information on how many cells the target is occupying.

In order to produce more detailed information, both as input and output, for the neural networks, segmentation will be the primary goal of both deep learning models implemented in this work. Therefore, the label associated with each RDM window will be a 16x16 ground truth mask with binary labels of 0 or 1. Examples of RDM windows and their corresponding labels are shown in Figure 3.4.

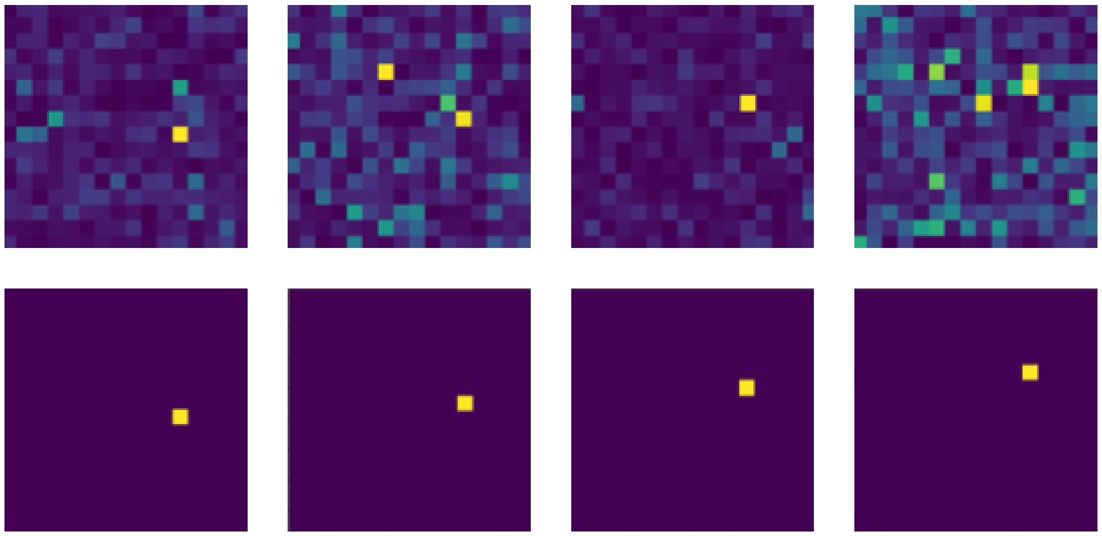
Figure 3.4: Example RDM window sequence and corresponding masks

3.2.4 Model Inputs

In order to provide sufficient training examples of both target-present and target-free windows, the original datasets windows were further broken down into an experimental dataset with equal target and non-target windows. This experimental dataset was split into training, validation, and testing subsets to perform cross-validation on the machine learning models. Training subsets were composed of 70% of the experimental windows, while the validation and testing subsets each consisted of 15% of the remaining windows. Table 3.2 shows the breakdown of each dataset used for training and testing at each SNR level for the staring radar class, and Table 3.3 shows the same information for the scanning radar class.

Table 3.2: Cross-validation dataset sizes for staring radar class

Dataset	**Total Training Windows**	**Total Validation Windows**	**Total Testing Windows**
30dB SNR	3270	700	700
10dB SNR	3488	747	747
3dB SNR	3414	731	731

Table 3.3: Cross-validation dataset sizes for scanning radar class

Dataset	**Total Training Windows**	**Total Validation Windows**	**Total Testing Windows**
30dB SNR	3294	705	705
10dB SNR	3254	697	697
3dB SNR	3424	733	733

3.3 NoisyLSTM Approach

The first deep learning architecture chosen for the target detection task was the NoisyLSTM. NoisyLSTM is a convolutional LSTM originally designed to perform segmentation across several frames in videos (Wang et al., 2021). This architecture was chosen for several reasons. First, the inspiration behind the NoisyLSTM was to maximize segmentation accuracy in low-quality video inputs. Because of this, the model was designed to suppress low-resolution or noisy video input to detect fine-details that can be captured in a predicted segmentation mask. Another appealing aspect of this model is the fact that it considers several sequential samples as a single input for the model. Originally, this behavior mapped to processing several video frames as a single input in order to capitalize on similarities within the frames over time. This is similar to the target detection problem where the signal returned by a target is ideally fairly consistent across multiple radar returns, but noise will continuously fluctuate over time. Rather than feed several video frames as input into the NoisyLSTM, sequences of RDM windows across multiple time steps were fed into the model instead. This behavior is visualized in Figure 3.5.

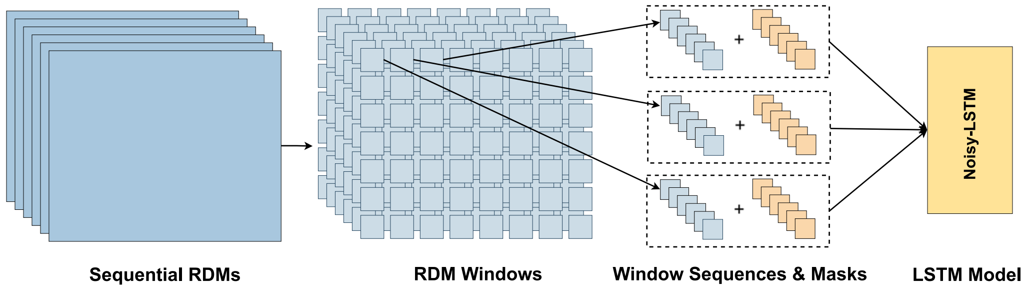

Figure 3.5: Data pipeline for NoisyLSTM model

The overall NoisyLSTM architecture is shown in Figure 3.6. The first stage of the NoisyLSTM is a feature extractor module based on the popular ResNet-101 architecture (He et al., 2016). ResNet-101 is a 101-layer CNN often used as a backbone for several computer vision architectures. The second stage of the NoisyLSTM architecture is a convolutional LSTM module. The convolutional LSTM follows the same general pattern as traditional LSTMs and other RNNs, but with the addition of several convolutional layers before the LSTM cell. Specifically, each sequential input and its corresponding hidden and cell states undergo a 2-D convolutional operation before being fed into the LSTM cell. This procedure is shown in Figure 3.7. The output of the convolutional LSTM module is then fed into two more convolutional layers with batch normalization and a rectified linear unit (ReLU) activation function before producing a segmentation mask prediction.

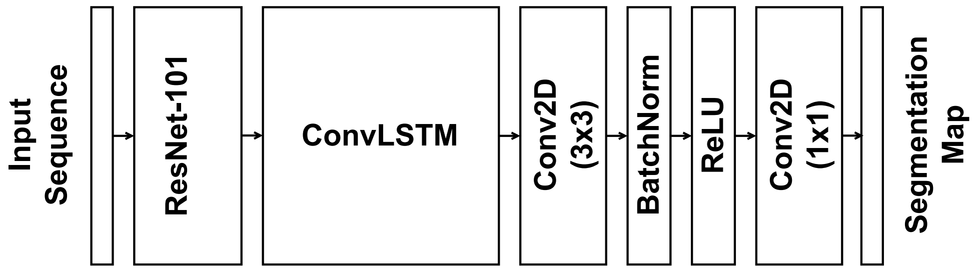

Figure 3.6: NoisyLSTM architecture

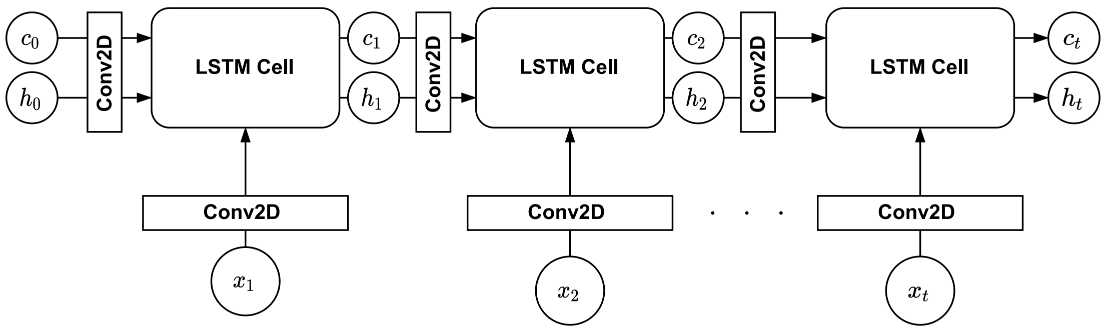

Figure 3.7: Convolutional LSTM overview

Batch normalization is a common supplemental layer used for stability during training. In many cases, individual samples in training data, including the input RDMs used in this work, may cover varying ranges of values. Batch normalization layers normalize the input data to a mean of zero and standard deviation of one to provide a uniform scale and ensure all input data covers the same range of numerical values. ReLU layers are also a common component of many neural network architectures. ReLU is an activation function that is defined as

$$R(x) = max(0, x)$$

for each element x in the input matrix being passed to the ReLU layer. ReLU is a computationally simple function that can significantly decrease training time by only activating a certain number of neurons at once.

Despite initial inspirations for this architecture, the NoisyLSTM proved to not be the best applicable model for this task. Despite multiple attempts to adapt the NoisyLSTM for target detection, initial training and testing runs demonstrated that the model was incapable of identifying target signatures over time. In most cases, the NoisyLSTM quickly converged during training to only predict segmentation masks with no targets present. This may imply that the initial CNN feature extraction module failed to capture any unique feature characteristics of the target signatures before the LSTM was tasked with tracking these features over time. Several attempts were made to modify the original NoisyLSTM architecture to be better-suited for the target detection problem. These attempted modifications included testing several different feature extraction modules, simplifying the architecture to its core CNN-LSTM component, and tuning various training parameters such as sequence length, batch size, and learning rate. However, these attempts were unsuccessful, and another deep learning architecture was tested using the knowledge gained from the failed NoisyLSTM approach.

3.4 U-Net Approach

Based on the results of the initial NoisyLSTM approach, a second model, U-Net (Ronneberger et al., 2015), was implemented for the target detection task. This model was originally created for biomedical image processing tasks where little training data may be available and the resulting segmentation mask must capture fine, pixel-level details in low-quality images. The high resolution of the output feature map stands out as a beneficial connection to the radar detection problem where target signatures may only exhibit minute differences amongst the noisy background. The U-Net architecture is composed of several blocks, referred to here as double convolutional blocks, as shown in Figure 3.8. Each block contains repeated applications of two convolutional layers with a kernel size of 3x3, two batch normalization layers, and two ReLU layers.

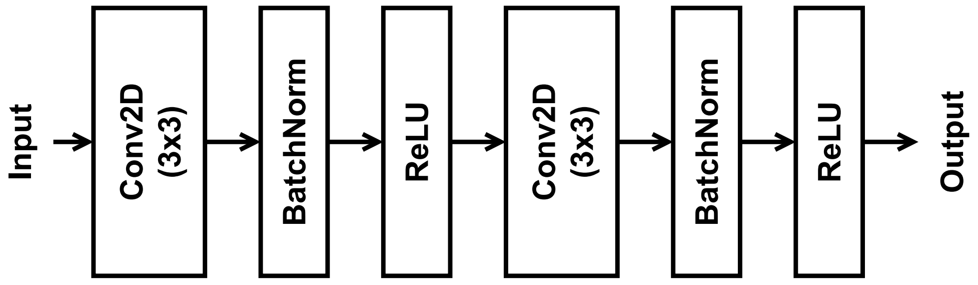

Figure 3.8: Double convolution block for U-Net architecture

The double convolutional blocks are further organized into two branches that make up the overall U-Net architecture: a contracting branch and an expansive branch. The combination of these two branches forms the overall architecture shown in Figure 3.9. The contracting branch (left) is structured similarly to a traditional CNN. In the contracting path, the output of each double convolutional block is downsampled using a 2x2 max pooling operation and the number of feature channels is doubled. The expansive path follows a similar pattern except that a 2x2 convolution is applied to the output of each double convolutional block to halve the number of feature channels.

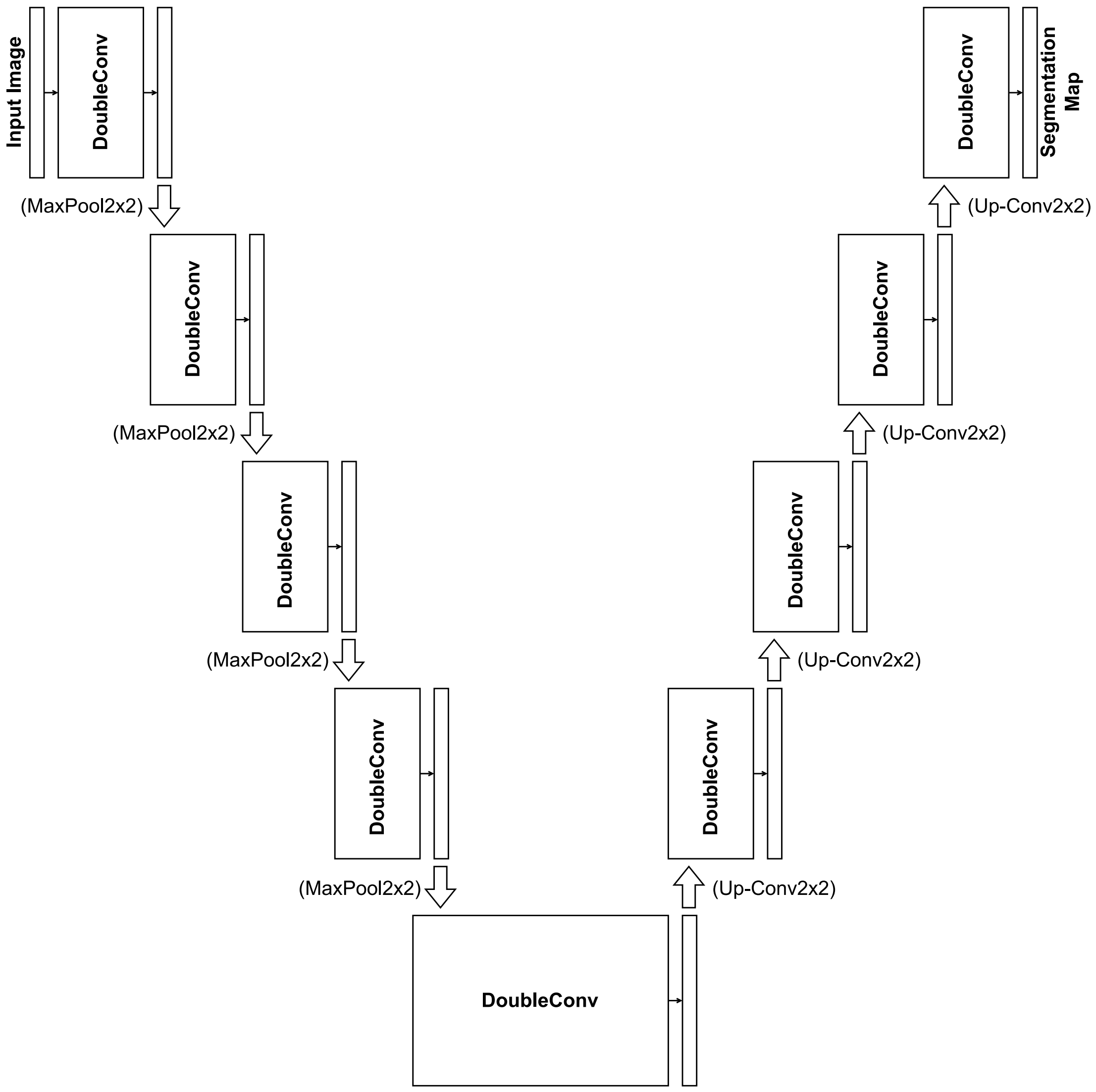

Figure 3.9: U-Net architecture diagram

Although the U-Net architecture is not reliant on time-series data, an additional preprocessing step inspired by Gusland et al. (2020) and Wang et al. (2021) was added to incorporate and exploit the temporal aspect of the range-Doppler data tested. The NoisyLSTM, which was designed to be tested with and without an LSTM layer, must be capable of accepting both sequential and non-sequential input data. In order to compensate for this, the NoisyLSTM model condenses sequential data through concatenation in order to format it into a shape acceptable to the 2-D convolutional layers while still retaining the temporal element of the data to be unrolled and processed at each timestep in the LSTM layer (Wang et al., 2021). This concatenation step is very similar to the point-wise addition of RDMs discussed earlier (Gusland et al., 2020) where RDMs across several timesteps are condensed into a 2-D image. Therefore, the model implemented for this task "stacks" multiple RDMs to create a new single input by taking the point-wise addition of four RDMs across four timesteps to create a new condensed map.

Initial tests with the modified U-Net architecture showed significant improvements over the original NoisyLSTM model tested and was chosen as the base model evaluated during training and testing.

3.5 Performance Metrics

3.5.1 Correct Detections

Before detailing the metrics used to analyze the performance of the U-Net model, it is important to first define what qualifies as a target detection based on the model output. The output of the U-Net model is a series of predicted segmentation masks for each RDM window given as input. In addition to the original input window, a corresponding ground truth mask is provided in order to compare the prediction to the expected output. Detections are determined by bounding boxes drawn around groupings of target cells (i.e. mask subsections where pixel values equal 1) with no padding. The bounding boxes drawn around target cell clusters in the ground truth mask are considered ground truth detections, and bounding boxes drawn around target cell clusters in the predicted mask are considered predicted detections. For every predicted detection, it is considered a correct detection if its bounding box overlaps with a ground truth bounding box. If not, it is considered a false alarm.

3.5.2 Probability of Detection and Probability of False Alarm

P_D and P_{FA} are two common metrics used for target detection in radar tasks, and will be included in the following performance analysis of the U-Net model. However, in traditional Neyman-Pearson-based radar detection approaches, P_{FA} is a predetermined value used to set a detection threshold to maximize P_D (Kay, 2009). In the case of this and other machine learning approaches (Baird et al., 2020) (Gusland et al., 2020), both metrics are based solely on the output of the machine learning models. For this work, P_D is defined as

$$P_D = \frac{N_C}{N_T}$$

where N_C is the total number of correct target detections and N_T is the total number of targets present. P_{FA} is defined as

$$P_{FA} = \frac{N_{FA}}{N_D}$$

where N_{FA} is the total number of false alarms and N_D is the total number of predicted target detections. It is important to note that this definition of P_{FA} differs from traditional signal processing approaches that calculate false alarm rates on a per pixel basis. For the purposes of this work, N_D is the total number of predicted targets identified by the deep learning models, i.e. the total number of predicted target bounding boxes. Therefore, P_{FA} in this case is representative of detections as a whole rather than on a pixel-by-pixel basis.

3.5.3 Incorrect Cell Predictions

While P_D and P_{FA} are the primary metrics of interest from a radar perspective, it is also important to consider how accurate the resulting U-Net segmentation mask is when compared to the ground truth mask provided to the model as a label. For this reason, the number of incorrect cells per RDM window is considered in tandem with P_D and P_{FA}. This metric is calculated by comparing a predicted RDM window and its ground-truth label to determine how many cells they differ by. This will include both cells that have been incorrectly labeled as targets (false alarm cells) and cells that have been incorrectly labeled to have no target when a target is actually present.

Figure 3.10 demonstrates three examples of why it is important to consider P_D, P_{FA}, and the number of incorrect cells collectively - especially under circumstances where the target occupies a very small region of the RDM window. In the leftmost example, the predicted mask includes a single cluster of predicted target cells that overlaps with the ground truth target, resulting in a $P_D = 1$ and $P_{FA} = 0$. However, the large target area also results in 46 incorrect cells when the predicted mask is compared to the ground truth label. The center example demonstrates the inverse problem. In this case, only 4 cells are incorrect, but the proposed target region shares no overlap with the ground truth target so that $P_D = 0$ and $P_{FA} = 1$. The rightmost example showcases the ideal prediction scenario, an output mask that highlights the correct target location within a reasonable localized region.

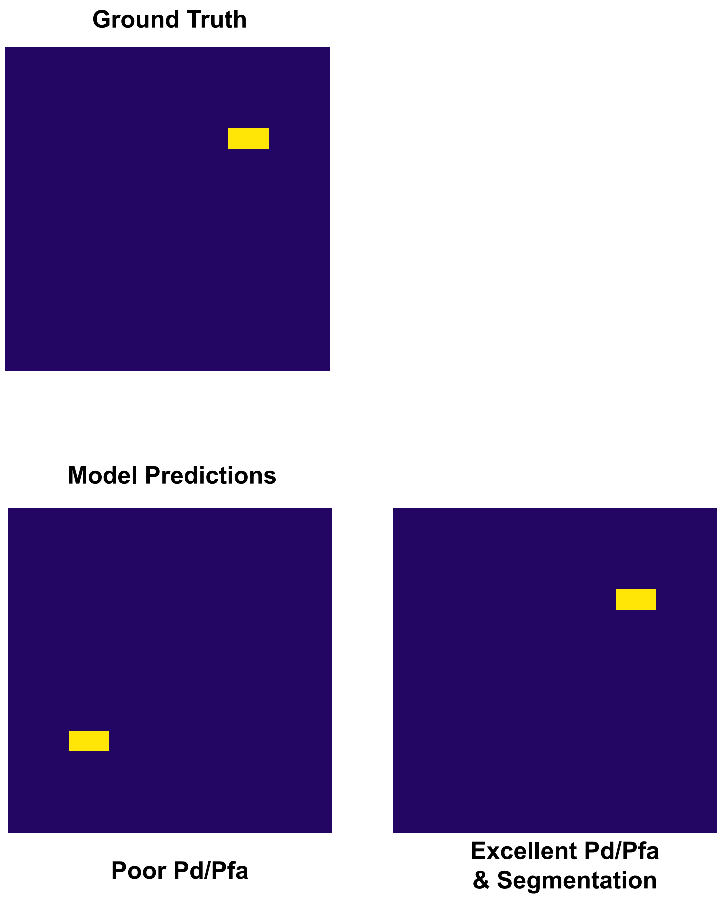

Figure 3.10: Sample of varying metric evaluations

3.5.4 Cross Entropy Loss

The loss function in this application will calculate the error for predictions made during the training stage by comparing them to their expected output. The U-Net model utilizes a combination cross-entropy and softmax loss function that places higher weights on labels distinguishing the edge between background and objects (Ronneberger et al., 2015). The softmax function for this binary classification problem is defined as

$$p_k(x) = \frac{e^{a_k(x)}}{\sum_{k'=1}^{2} e^{a_{k'}(x)}}$$

where $a_k(x)$ is the activation in each feature channel k at each pixel position $x \in \Omega$, and $p_k(x)$ is the approximated softmax value. The value of $p_k(x)$ should be closer to 1 for the class with the highest predicted probability. The cross entropy loss function is then utilized to penalize the predicted class at each pixel value using

$$E = \sum_{x \in \Omega} w(x) log(p^*(x))$$

where $w(x)$ is the current weight map and $p^*(x)$ is the expected label at pixel $x \in \Omega$. A low loss value indicates little variability between the predicted and expected output, while a high low value implies the opposite.

Chapter 4

Results

In order to test the U-Net architecture on the RDM windows, the original architecture codebase provided by (Ronneberger et al., 2015) was modified and adapted to the staring and scanning radar datasets described in Section 3. A majority of these changes were made using PyTorch, an open source framework for designing and testing machine learning models. Using PyTorch, the input data, training parameters, and evaluation metrics could be updated to fit this target detection application while keeping the core U-Net architecture the same. A detailed overview of the U-Net architecture created with PyTorch is included in Table 4.1 Evaluation of the applied U-Net model was then divided into two stages: training and testing.

4.1 U-Net Training

Training sessions were conducted on the training and validation datasets for both staring and scanning radar classes at 30dB, 10dB, and 3dB SNR. During training, the model iterates through each sample in the training dataset and feeds that

Table 4.1: Detailed architecture of implemented U-Net model

Operation	**Layer**	**Filters**	**Filter Size**	**Stride**
Conv Layer	Conv2D	64	(3x3)	(1x1)
(repeated twice)	BatchNorm	-	-	-
	ReLU	-	-	-
Pooling Layer	MaxPool	1	(2x2)	(2x2)
Conv Layer	Conv2D	128	(3x3)	(1x1)
(repeated twice)	BatchNorm	-	-	-
	ReLU	-	-	-
Pooling Layer	MaxPool	1	(2x2)	(2x2)
Conv Layer	Conv2D	256	(3x3)	(1x1)
(repeated twice)	BatchNorm	-	-	-
	ReLU	-	-	-
Pooling Layer	MaxPool	1	(2x2)	(2x2)
Conv Layer	Conv2D	512	(3x3)	(1x1)
(repeated twice)	BatchNorm	-	-	-
	ReLU	-	-	-
Pooling Layer	MaxPool	1	(2x2)	(2x2)
Conv Layer	Conv2D	1024	(3x3)	(1x1)
(repeated twice)	BatchNorm	-	-	-
	ReLU	-	-	-
Up-Conv Layer	ConvTranspose2D	512	(2x2)	(2x2)
Conv Layer	Conv2D	512	(3x3)	(1x1)
(repeated twice)	BatchNorm	-	-	-
	ReLU	-	-	-
Up-Conv Layer	ConvTranspose2D	256	(2x2)	(2x2)
Conv Layer	Conv2D	256	(3x3)	(1x1)
(repeated twice)	BatchNorm	-	-	-
	ReLU	-	-	-
Up-Conv Layer	ConvTranspose2D	128	(2x2)	(2x2)
Conv Layer	Conv2D	128	(3x3)	(1x1)
(repeated twice)	BatchNorm	-	-	-
	ReLU	-	-	-
Up-Conv Layer	ConvTranspose2D	64	(2x2)	(2x2)
Conv Layer	Conv2D	64	(3x3)	(1x1)
(repeated twice)	BatchNorm	-	-	-
	ReLU	-	-	-
Conv Layer	Conv2D	2	(3x3)	(1x1)

sample as input into the network. It is also important to note that this is a form of supervised learning in which the model has access to both the training samples and corresponding labels (in this case, the expected segmentation masks) for use during the entire training stage. After the model makes a prediction on the training sample, the loss function is applied to compute the difference between the predicted and expected output, and the model's weights are updated based on the calculated loss value. Another important step in the training process is the validation step. After the model has iterated through all samples provided in the training dataset, it then continues to iterate through the validation dataset as well. The same loss-calculation procedure is followed for the validation step; however, the calculated loss value is not used to update the model weights. Instead, the validation step is used to monitor how well the model is able to generalize. The results of the validation step provide the best insight on whether or not overfitting has occurred. In the case where the model produces a low training loss and a high validation loss, it is possible overfitting has occurred and the model is no longer capable of generalizing to new or unseen data.

Table 4.2 lists a few key training parameters that were tuned for training the original U-Net architecture on the RDM window datasets.

Table 4.2: Modified training parameters

Parameter	**Value**	**Definition**
Epochs	35	Number of full iterations over the training and validation datasets
Batch Size	64	Number of training samples given to the model at once
Learning Rate	0.001	Rate at which model weights are updated

4.1.1 Training and Validation Loss

Figure 4.1 shows the loss calculated at each epoch on both the training and validation datasets across all SNR classes for the staring radar data. All three SNR classes followed a very similar loss pattern over 35 epochs. The 30dB SNR class achieved the lowest training and validation losses, both achieving loss values as low as 0.01. This is to be expected as targets within this class are the easiest to identify among noise. The 10dB SNR class achieved the second lowest training and validation losses at 0.19 and 0.22 respectively. Lastly, the 3dB SNR class produced the highest training and validation loss values at 0.49 and 0.50. There are two important behaviors to note within these results. First, all three classes experienced a sharp drop in loss during early epochs, and eventually stabilized over time. This indicates that the model was actively learning from poor predictions made early on in training. The 30dB and 3dB SNR classes show very little decrease after the initial drop in loss. In the case of the 30dB SNR class, this is most likely because the model quickly converges and is capable of easily identifying targets within each RDM window. The 3dB SNR class most likely exhibits this behavior for the opposite reason, it is struggling to learn identifiable target features and make accurate predictions. Further evidence for both of these behaviors is discussed in subsequent sections. The second important behavior to note across all three classes is that the training and validation losses are very similar throughout training. This is an early indicator that the model was capable of generalizing well and was not overfitting to the training data.

Figure 4.2 shows the training and validation loss calculated at each epoch for the scanning radar class. While there are similarities among the loss patterns for this training class as well, it is clear that the U-Net model did not perform

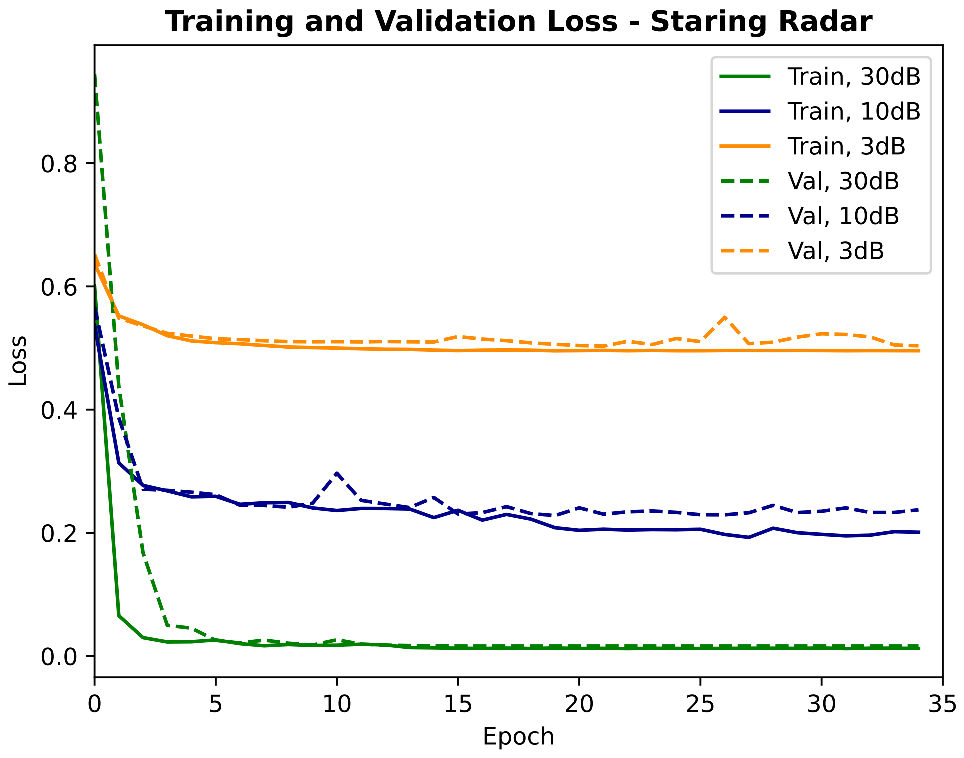

Figure 4.1: Training and validation loss for staring radar over 35 epochs

as well on the scanning radar data at lower SNR values when compared to the staring radar results. The loss values for the 30dB SNR class were very similar to the staring radar class, again achieving training and validation loss values as low as 0.01. Conversely, the training and validation loss for the 10dB and 3dB SNR classes were noticeably higher for the staring radar class. The 10dB SNR class reached a training loss of 0.45 and a validation loss of 0.49, and the 3dB SNR class reached a training loss of 0.56 and a validation loss of 0.59.

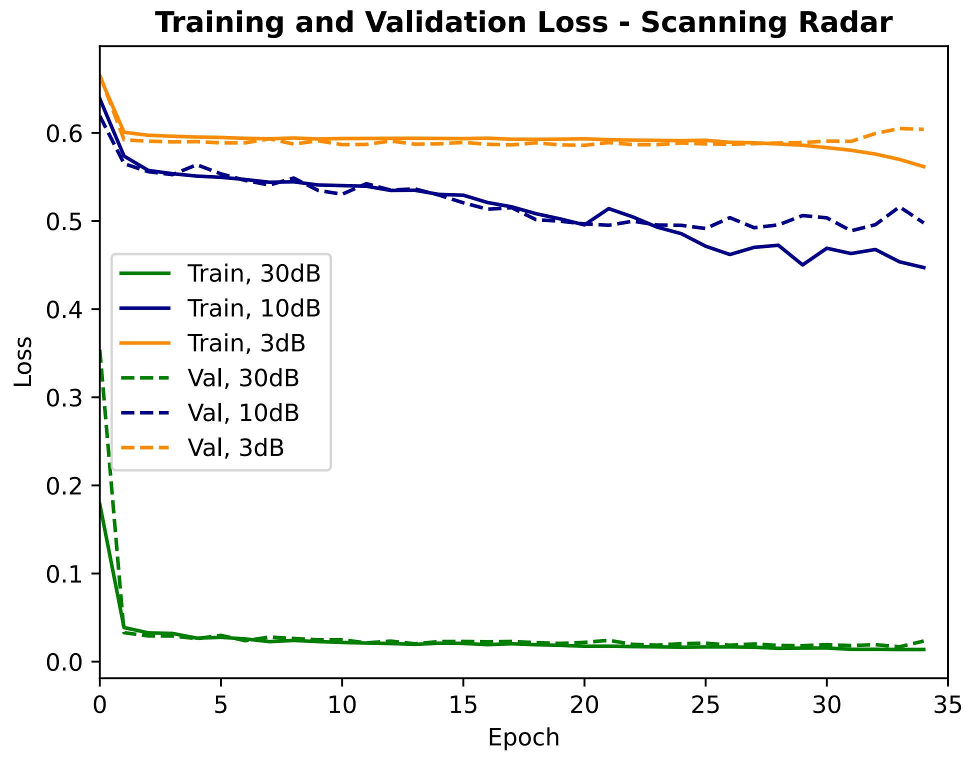

Figure 4.2: Training and validation loss for scanning radar over 35 epochs

4.1.2 Probability of Detection and False Alarm

In addition to monitoring loss, P_D and P_{FA} metrics were calculated on the results of the training dataset to monitor the performance of the model throughout the training stage. The P_D and P_{FA} results for the staring radar class are shown in Figure 4.3. Each model's results fall in-line with what is to be expected based on the training losses shown in Figure 4.1. The 30dB SNR class quickly converged to an optimal result, achieving a P_D of 1 and P_{FA} of 0. The 10dB SNR class also performed well and showed steady improvements throughout training. The best P_D and P_{FA} achieved for this class were 0.97 and 0.11 respectively. The 3dB SNR class again produced the worst results with a P_D that only reached 0.29 while the P_{FA} remained high at 0.71.

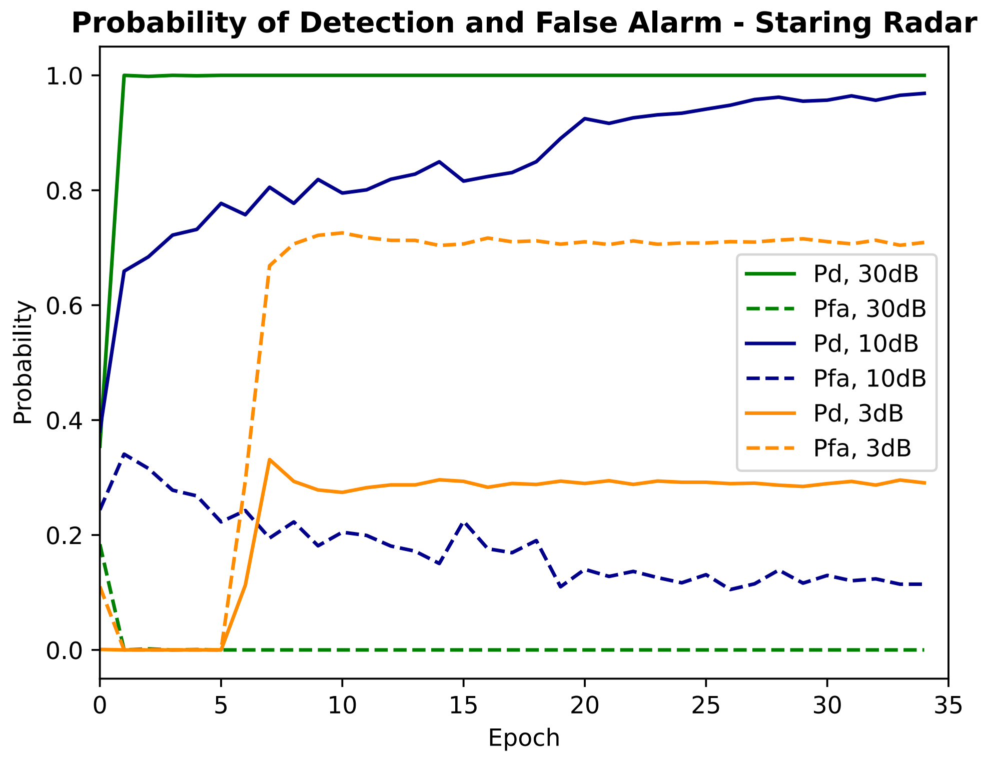

Figure 4.3: Probability of detection and false alarm for staring radar over 35 epochs

Figure 4.4 shows the P_D and P_{FA} metrics calculated during training for the scanning radar class. Again, these results fall in-line with what is to be expected based on the training losses shown in Figure 4.2. The 30dB SNR class quickly converged to an optimal result and achieved a P_D of 0.99 and P_{FA} of 0.01. Furthermore, both of the lower SNR classes struggled to make consistent and accurate predictions throughout the training process. The best P_D and P_{FA} metrics achieved for the 10dB SNR class were 0.39 and 0.60 respectively. The 3dB SNR class also produced poor results with a P_D that only reached 0.26 with a correspondingly high P_{FA} of 0.79.

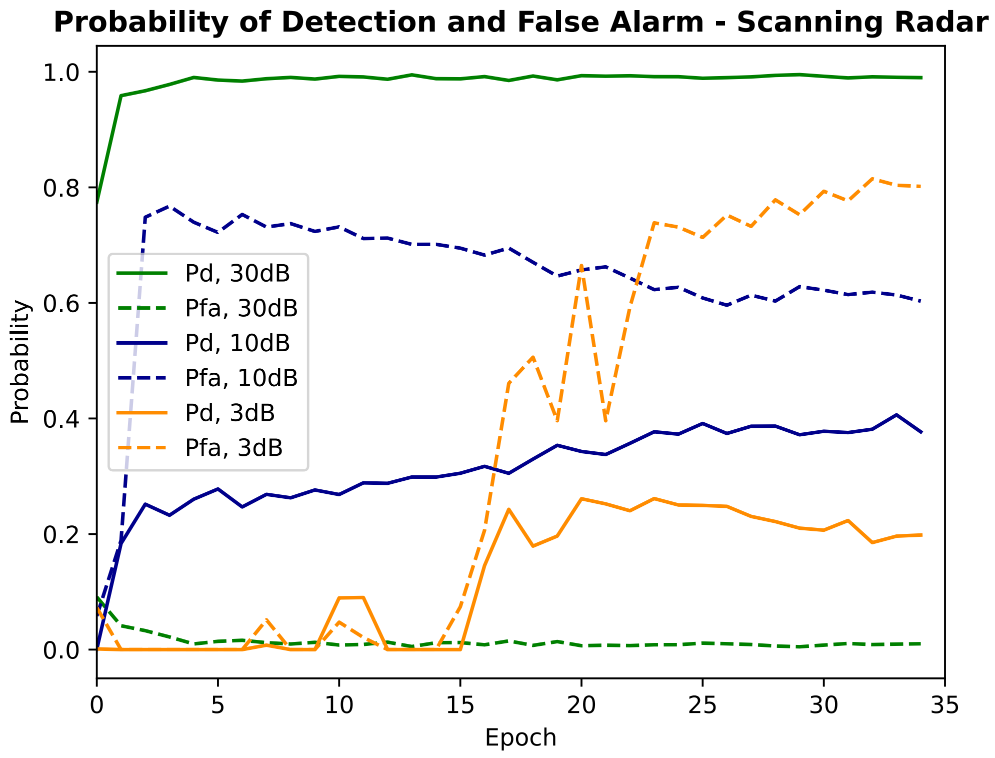

Figure 4.4: Probability of detection and false alarm for scanning radar over 35 epochs

4.1.3 Incorrect Cell Predictions

The last metric measured during training was the average number of incorrectly predicted cells per 16x16 RDM window. Figure 4.5 shows the results of this metric on the staring radar class. The results across all three classes were positive, where the 30dB, 10dB, and 3dB classes all averaged 0.05, 0.98, and 1.87 incorrect cells respectively. This is positive in the fact that the predicted segmentation masks for the 30dB SNR class rarely differed by even a pixel when compared to the ground truth mask. Similarly, the predicted masks for the 10dB and 3dB SNR classes often only differed by 1-2 incorrect cells per window.

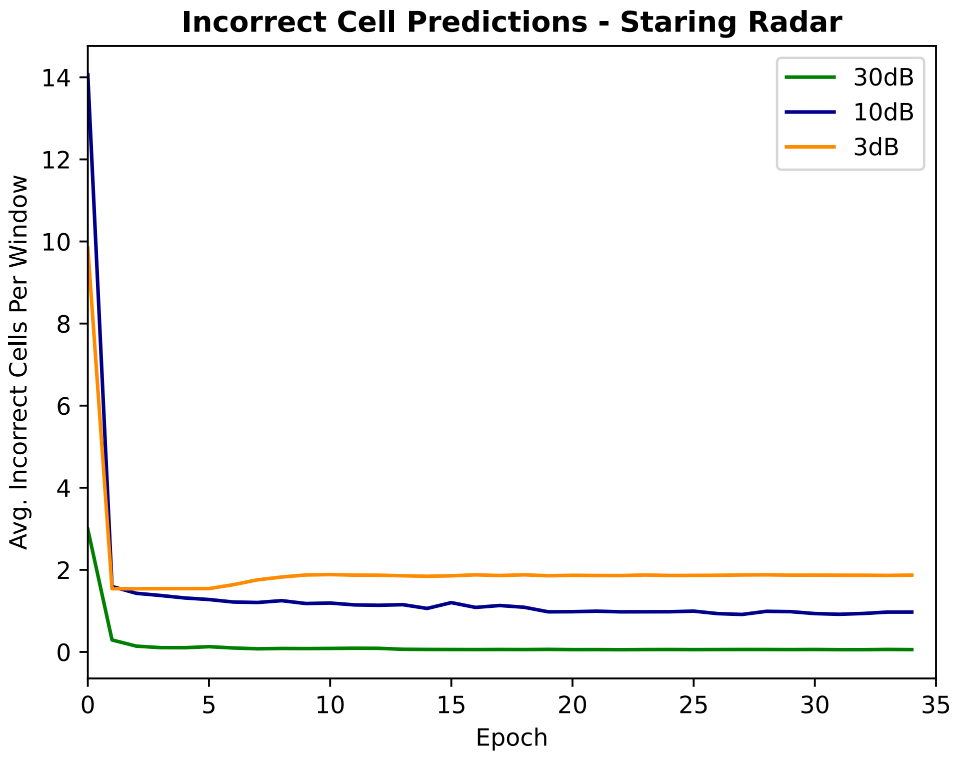

Figure 4.5: Average incorrect cell predictions for staring radar over 35 epochs

Figure 4.6 shows the average number of incorrectly predicted cells per window in the scanning radar class. Unsurprisingly, these results also show a noticeable difference in performance between the 30dB SNR results and the lower 10dB and 3dB SNR results. Similar to the staring radar results, the 30dB SNR class rarely misclassified pixels in the predicted segmentation masks, resulting in an average 0.19 incorrect cells per window. Unfortunately, the scanning radar class did differ from the staring results for the 10dB and 3dB SNR windows. The 10dB SNR class averaged 3.30 incorrect cells per RDM window, and the 3dB SNR class averaged 5.10 incorrect cells per RDM window. While 3-5 cells still only represents only a small portion of each 16x16 segmentation mask, it is important to note the difference in performance of the U-Net model on the staring and scanning radar data.

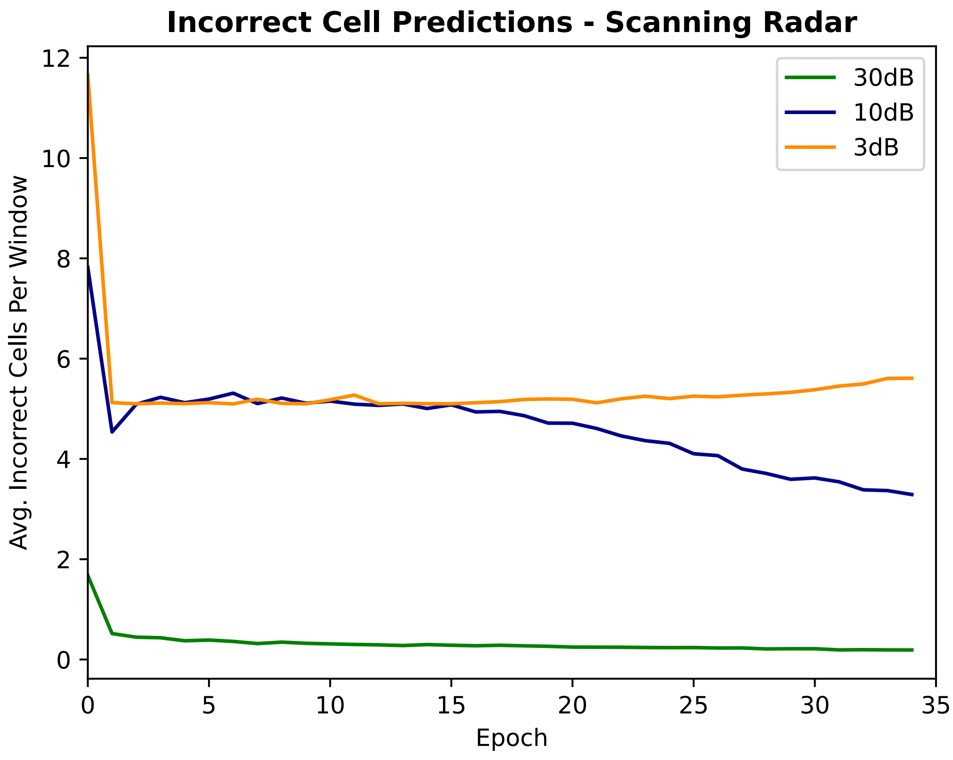

Figure 4.6: Average incorrect cell predictions for scanning radar over 35 epochs

4.2 U-Net Testing

After training was completed, the weights from the model with the best validation performance were used for testing. The testing stage operates very similarly to the validation step implemented during training. The trained model with pre-loaded weights iterates through all data samples in the test dataset. Unlike training, the model is not given the expected output labels during testing, but the labels are used afterwards to calculate the performance metrics shown in Table 4.3 and Table 4.4 for the staring and scanning radar classes.

Table 4.3: Test results for staring radar class

SNR Class	P_D	P_{FA}	Avg. Incorrect Cells
30dB	1.00	0.00	0.68
10dB	0.95	0.01	1.71
3dB	0.36	0.60	1.69

Table 4.4: Test results for scanning radar class

SNR Class	P_D	P_{FA}	Avg. Incorrect Cells
30dB	0.99	0.01	0.21
10dB	0.36	0.64	5.29
3dB	0.17	0.78	4.95

4.2.1 30dB SNR Results

As expected, the U-Net model trained to detect targets at 30dB SNR performed the best, with a $P_D = 1$ and $P_{FA} = 0$ for the staring radar and a $P_D = 0.99$ and $P_{FA} = 0.01$ for the scanning radar. In this case, the model had no problem correctly identifying targets present in both test datasets. In the case of the staring radar class, the model never produced a target detection in a window region where a target was not present. In the case of the scanning radar class, missed detections and false alarms were also a very rare occurrence. Additionally, the average number of incorrect cells was very low for both classes, only 0.68 for the staring class and 0.21 for the scanning class, implying that not only was the model capable of identifying the presence of a target, but it was capable of localizing on its precise location within the RDM window with a high degree of accuracy.

4.2.2 10dB SNR Results

The trained U-Net models for targets at 10dB SNR showed the largest difference in performance between the staring and scanning radar classes. The U-Net model trained on the staring radar data produced promising results, with $P_D = 0.95$, $P_{FA} = 0.01$, and an average 1.71 incorrect cells per window mask. Although 0.01 is a relatively high P_{FA} for traditional target detectors using signal processing techniques, these results indicate using neural networks as a preprocessing step for more sophisticated target detectors may be a promising area of research. Once a model, U-Net for example, is trained on representative training data, the overhead for applying the model to new data drops significantly. Therefore, U-Net may serve as a valuable tool to identify regions where computationally complex signal processing techniques should be applied. Furthermore, the low incorrect cell average also indicates that the region highlighted as a possible target will be localized to a small subregion of the RDM window, further reducing the search space of traditional detection techniques.

Results on the scanning radar data were comparatively worse for the 10dB SNR targets, with $P_D = 0.36$, $P_{FA} = 0.64$, and an average 5.29 incorrect cells per window mask. Further research is necessary to understand the reasoning behind this performance drop. It is clear based on these initial results that differences in range-Doppler data and collection methods may require adjustments in the deep learning models before being used as a preprocessing step. These adjustments may include tuning training parameters, modifying layers within the neural network architecture, or applying additional preprocessing strategies to different forms of input data. This highlights the importance of testing deep learning methods on a variety of range-Doppler data. In many cases, neural networks will need to be tailored to the specific radar system and application, or additional testing after tuning may show improved results for new forms of data so that the models can generalize.

4.2.3 3dB SNR Results

While the performance of the U-Net models on 10dB SNR targets differed significantly between the staring and scanning radars, both radar classes struggled to make accurate predictions for targets at 3dB SNR. For the staring radar, the trained model produced poor detection and false alarm rates with $P_D = 0.36$ and $P_{FA} = 0.60$. The scanning radar model also produced poor detection and false alarm rates with $P_D = 0.17$ and $P_{FA} = 0.78$. Targets in this category often only occupy a single cell within the RDM. Although the average number of incorrect cell predictions is still low, the poor P_D and P_{FA} rates indicate that the model struggled to localize on the target's location within the RDM window. These results indicate that the target signatures within this SNR class did not exhibit enough unique spatial characteristics to make them distinguishable among noise. To apply this machine learning technique for targets at this SNR level, additional preprocessing steps must be examined to either further increase the SNR of these targets or increase the number of unique features that can be detected by neural networks.

4.3 Discussion

The training and testing results for the U-Net model showcased the benefits and limitations of this deep learning architecture. The results of the 30dB SNR class served as a proof-of-concept that the U-Net architecture could successfully be adapted to process range-Doppler radar data in order to perform target detection. The 10dB SNR class served as a more challenging example of weak target signals. For the staring radar class, the U-Net model was still capable of achieving a high P_D of 0.95 while maintaining a P_{FA} as low as 0.01. However, the U-Net model could not easily produce the same results on the scanning radar class, ultimately achieving a P_D of 0.36 and a P_{FA} 0f 0.64. Unsurprisingly, the lowest SNR class tested, 3dB, produced the worst results for both radar classes. It is clear additional preprocessing and data augmentation would be required to apply this architecture to increasingly small SNR values.

Other applications of machine learning to target detection in radar have also achieved high detection rates, but often at the expense of correspondingly high false alarm rates as well. For example, Wang et al. (2022) also proposed a CNN-based approach to detecting targets in range-Doppler maps that produced a perfect $P_D = 1$ on targets at 8dB and 12dB SNR, but a P_{FA} as high as 0.75 as well. Gusland et al. (2020) also proposed a CNN-based target detection approach that achieved a much lower $P_{FA} = 0.001$, but only reached a $P_D = 0.8$. While differences in input data, performance metrics, and training environments make it difficult to directly compare these approaches, the differences in recorded results do demonstrate the difficulty of the original problem and the trade-offs that are often made when balancing P_D and P_{FA}.

Chapter 5

Conclusions

In this work, two deep learning neural networks were applied towards target detection in radar systems. Both models were tested on simulated range-Doppler data in hopes of identifying targets in increasingly-low SNR scenarios. The first model tested, NoisyLSTM, was chosen in the hopes of exploiting the sequential nature of the input range-Doppler data using an RNN. Although this idea seemed promising at first, the NoisyLSTM was unable to distinguish target signatures from noisy signals. The second model tested, U-Net, proved to be much more successful. This modified CNN architecture was still capable of incorporating temporal information within the input data by stacking, or taking the point-wise summation, of several sequential RDMs to amplify targets' consecutive signals over time and create a higher target return, often with unique spatial features such as streaking, within a single input sample. While the results produced by the U-Net model are not capable of replacing traditional signal processing techniques entirely, the results do indicate that the U-Net architecture may serve as a beneficial preprocessing step to reduce the search space for detectors and reserve resources.

Chapter 6

Future Work

There are multiple areas of interest to be further explored within the work presented in this thesis. The most obvious step for future work would be incorporate other existing SNR-enhancing techniques in an attempt to improve the model's performance on SNRs as low as 3dB. Additional work must also be done to analyze the difference in performance between the staring and scanning radar classes at 10dB SNR. Understanding the underlying factors behind this imbalance will provide insight on how to improve either the U-Net model or preprocessing steps to hopefully generalize the architecture to perform well on a wider variety of data.

Future work should also consider incorporating more realism into the radar simulation used to generate the training, validation, and test datasets, or to create training datasets using real range-Doppler data as well. Adding more realism into the training data will most likely affect the prediction quality of the network. On one hand, clutter or equipment interference may add additional noise to the range-Doppler data, resulting in more noise spikes and further masking the target signal. However, more realistic training data may actually help the U-Net model make more accurate predictions if the model can capitalize on repeating feature patterns in targets that are not currently modeled in the simulation described in Section 3.1. One possible example of this would be testing if sidelobe patterns produced by targets may actually help localize on the original target's location itself. Another interesting area for future work would be to extend the U-Net model to include multi-class detection problems. In this case, the model would be capable of not only identifying targets within RDMs, but classifying all detected targets into various categories.

The work presented in this thesis highlights several interesting areas of research centered around target detection, and it is clear there are still several areas of research that have yet to be explored. Existing applications of machine learning for radar tasks have shown exciting and promising results, but continued work must be done to analyze the potential for these applications in real-world scenarios.

Bibliography

Saad Albawi, Tareq Abed Mohammed, and Saad Al-Zawi. Understanding of a convolutional neural network. In *2017 International Conference on Engineering and Technology (ICET)*, pages 1–6, 2017. doi: 10.1109/ICEngTechnol.2017.8308186.

Marco Altmann, Peter Ott, Nicolaj C. Stache, and Christian Waldschmidt. Learning dynamic processes from a range-doppler map time series with LSTM networks. In *2019 16th European Radar Conference (EuRAD)*, pages 13–16, 2019.

Zachary Baird, Michael K. Mcdonald, Sreeraman Rajan, and Simon J. Lee. A CNN-LSTM network for augmenting target detection in real maritime wide area surveillance radar data. *IEEE Access*, 8:179281–179294, 2020. doi: 10.1109/ACCESS.2020.3025144.

Zongyong Cui, Xiaoya Wang, Nengyuan Liu, Zongjie Cao, and Jianyu Yang. Ship detection in large-scale sar images via spatial shuffle-group enhance attention. *IEEE Transactions on Geoscience and Remote Sensing*, 59(1):379–391, 2021. doi: 10.1109/TGRS.2020.2997200.

Ian J. Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, Cambridge, MA, USA, 2016.

Klaus Greff, Rupesh Srivastava, Jan Koutník, Bas Steunebrink, and Jürgen Schmidhuber. LSTM: A search space odyssey. *IEEE transactions on neural networks and learning systems*, 28, 03 2015. doi: 10.1109/TNNLS.2016.2582924.

Emanuele Grossi, Marco Lops, and Luca Venturino. A novel dynamic programming algorithm for track-before-detect in radar systems. *IEEE Transactions on Signal Processing*, 61(10):2608–2619, 2013. doi: 10.1109/TSP.2013.2251338.

Daniel Gusland, Sigmund Rolfsjord, and Børge Torvik. Deep temporal detection - A machine learning approach to multiple-dwell target detection. In *2020 IEEE International Radar Conference (RADAR)*, pages 203–207, 2020. doi: 10.1109/RADAR42522.2020.9114828.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016. doi: 10.1109/CVPR.2016.90.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9:1735–80, 12 1997. doi: 10.1162/neco.1997.9.8.1735.

Steven M. Kay. *Fundamentals Of Statistical Processing, Volume II: Detection Theory*. Prentice-Hall signal processing series. Pearson Education, 2009. ISBN 9788131729007.

J.D. Kelleher. *Deep Learning*. MIT Press essential knowledge series. MIT Press, 2019. ISBN 9780262354899.

Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1(4):541–551, 1989. doi: 10.1162/neco.1989.1.4.541.

Tim O'Shea, Tamohgna Roy, and T. Charles Clancy. Learning robust general radio signal detection using computer vision methods. In *2017 51st Asilomar Conference on Signals, Systems, and Computers*, pages 829–832, 2017. doi: 10.1109/ACSSC.2017.8335463.

Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 779–788, 2016. doi: 10.1109/CVPR.2016.91.

Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-Net: Convolutional networks for biomedical image segmentation. In Nassir Navab, Joachim Hornegger, William M. Wells, and Alejandro F. Frangi, editors, *Medical Image Computing and Computer-Assisted Intervention (MICCAI) 2015*, pages 234–241, Cham, 2015. Springer International Publishing. ISBN 978-3-319-24574-4.

Bowen Wang, Liangzhi Li, Yuta Nakashima, Ryo Kawasaki, Hajime Nagahara, and Yasushi Yagi. Noisy-lstm: Improving temporal awareness for video semantic segmentation. *IEEE Access*, 9:46810–46820, 2021. doi: 10.1109/ACCESS.2021.3067928.

Chenxing Wang, Jiangmin Tian, Jiuwen Cao, and Xiaohong Wang. Deep learning-based uav detection in pulse-doppler radar. *IEEE Transactions on Geoscience and Remote Sensing*, 60:1–12, 2022. doi: 10.1109/TGRS.2021.3104907.