Applying Machine Learning Techniques to Radar Systems

Ivan Naumovski, inau@itu.dk
Supervision by Kasper Støy

June 1, 2017

Abstract

Analysing RADAR data is a non-trivial task due to the nature of the sensory data obtained, which is usually noisy, cluttered or messy in some other way. Extracting useful information in the messy data has been of utmost importance since the origin of RADAR.

The goal is to examine whether the popular neural networks can perform just as well as a baseline function in extracting information from RADAR data - the claim is that the networks ought to as they handle noisy data well. To investigate whether the networks can achieve better performance than the baseline, one needs to generate diverse RADAR data and prepare it for use with networks.

The data generation study is done with scalability in mind, such that methods for systematically defining experimental setups are produced. Finally different types of input scaling have been applied to the data to investigate what impact they do to the precision of the different networks.

In addition to input scaling, different network topologies have been explored. The indication is that the topologies, when restricted to only using two layers, do not seem to matter too much, whereas the input scaling does.

The networks do exhibit better performance than the baseline function on noisy data, which was expected.

Contents

1 Introduction

This project aims to investigate the inner workings of radio detection and ranging more commonly known under its popular name RADAR, and whether it is possible to apply machine learning techniques to analyse the response signals.

Discoveries about electromagnetic waves during the late 1800s and early 1900s by numerous scientist such as Heinrich Hertz and Guglielmo Marconi among others established the ground for detection systems. The discoveries of namely the reflection of EM waves on certain materials laid the foundation for some of the first RADAR systems.

One of the first to build detection systems was the scientist Christian Hüllsmeyer, which built a naval RADAR, however it was never really adopted by the naval institutions at the time. The RADAR technology, like alot of other technologies, rapidly grew during global conflicts, both during the first and the second world war.
The need for the conflicted parties to be able to efficiently detect and track both enemy and friendly targets fuelled the effort towards developing better systems.

Due to secrecy in most of the research programmes multiple people can be creditted for similar advancements and discoveries in roughly the same time periods.
At the time of writing so many different variations of these systems exist with different specialisations, ie. scanners used for speed measurement by law enforcers or surveillance RADARs at airports for airtraffic control(ATC), that it is out of scope to attempt and do a generalisation for all systems, instead i will focus on a standard system which i will present later.
To get an overview of how diverse systems are, some applications of RADAR systems have been listed in figure 1;

Coming from a software background one of the biggest challenges during this project has been understanding the differences of the systems and correctly configuring my test environments.

The following section 1.1 will present some of the obstacles encountered when designing RADAR systems and what approaches will be applied to overcome

Field	Application
Military	*air-defense, surveillance, high resolution imaging (SAR).*
Remote Sensing	*weather observation, planetary observation, ground probing, mapping of sea-ice for routing.*
Air Traffic Control	*Air Surveilance Radar (ASR), Air Route Surveilance Radar (ARSR), Airport Surface Detection Equipment (ASDE).*
Law Enforcement and Highway Safety	*speed measurements, collision avoidance, security systems(alarms).*
Aircraft Safety and Navigation	*weather avoidance radar, terrain avoidance, radio altimeter.*
Ship Safety	*collision avoidance.*
Space	*docking systems, mapping*

Table 1: Overview of different systems

them.

Section 2 will give an overview of how RADAR systems work and present the most common challenges encountered, followed by section 3 which will present how the interpretation and processing of signals can be handled.

1.1 Problems

Problems in RADAR arise quite frequently due to the sensing being influenced by multiple factors which can make analysis of the sensor data difficult. Some of the factors can be random, others can be constant. The point being that the influencing factors significantly decrease the ability to decide whether the sensed object is desired or not.

Factors such as noise, interference, line-of-sight, clutter and jamming heavily impact systems.

The *noise* factor can be both internal and external noise, and is usually referring to thermal noise which is always present in electronics.

Interference covers signals that are detectable by the RADAR, but not of interest to the current task at hand.

In contrast *clutter* is referring to targets that are detectable in the radio frequency of the system, but not interesting to the target. Consider elements like dust, tall buildings, chaff.

Interference is disruption on signal level, clutter is disruption in regards to responses.

Some of these limiting factors can be handled by configuring the RADAR system, which is why we focus on the factors external to the system, such as clutter. The end goal is to train a neural network to do basic detection.

Initially i will explore the inner workings of RADAR and investigate how to replicate systems at signal level. In addition to the above, the production of data in bulk will be investigated as this will be required later.

The next tier of exploration is in digital signal processing(DSP), during which some standard ways of processing RADAR data will be presented. This knowledge will help during the decision making for the machine learning exploration in section 5.

When applying machine learning techniques to predict RADAR data, the main investigation will be whether using a neural network is able to perform just as well as baseline processing functions for determining direction. The claim is that due to the networks exhibiting desirable features such as handling noisy data quite well, these are suited for applying to raw sensory data from RADAR systems.

2 Exploring RADAR and DATA Generation

This section aims to present the inner workings of RADAR, followed by the reasoning behind the decisions made when doing the data generation in section 2.2. The main sources of information have been *'Introduction to RADAR systems'* by M. I. Skolnik[12, Ch. 1,2,3] and the online resource *'RADAR tutorial'* by Christian Wolff [14].

2.1 RADAR System Basics

RADAR sensing as a concept is rather straightforward, it relies on the knowledge about electromagnetic waves and their propagation. Electromagnetic waves travel at the speed of light which is approximated as $300 \cdot 10^6 m/s$ - this information is what makes ranging possible, however without any notion of direction it is not possible to determine origin relative to the receiving end of the system.

The simplest way to explain RADAR is sensing of electromagnetic waves to map the environment relative to the receiving end of the system. Similar to the effects of dropping a pebble into still water, waves will travel outwards from the initial impact point until they are reflected back from some surface. The reflection from the new impact point will then be weaker, since some energy will be lost during the propagation of the wave, resulting in lower energy reflected back towards the origin point.

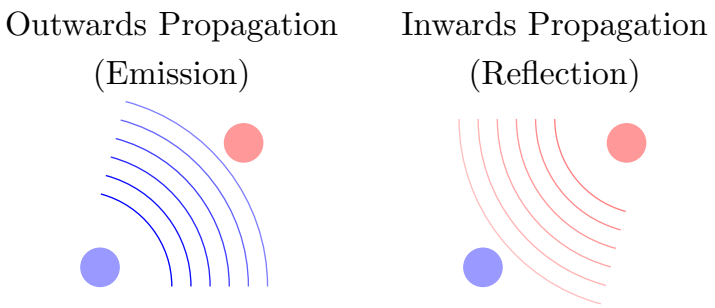


Figure 1: Example of transmission and reflection of a EM signal

These physical phenomena make up the foundation of RADAR systems. In short a RADAR system should be able to send and receive EM waves to replicate the above phenomena. This requires 3 key components an antenna, a transmitter and receiver.
The antenna is required by both the transmitter and receiver, systems can have multiple antennae. The transmitter is designed to send out signals whereas the receiver is designed to do the opposite.
The above with 3 components holds for the category of active RADAR systems, for passive systems only an antenna and a receiver are required. The passive systems mentioned can be airplane transponders posting their locations to a listener, in this case the targets themselves are providing the location and additional information, rather than the system attempting to bounce signals off the objects.

The previous section describes the wave propagation of RADAR systems in an informal way, the same can be explained by the following equation known as the RADAR equation:

$$P_r = \frac{P_t \cdot G_t \cdot A_r \cdot \sigma \cdot F^4}{(4\pi)^2 \cdot R_t^2 \cdot R_r^2} \tag{1}$$

Which states that the returned power from a reflection is equal to the emitted energy evenly distributed over the surface of two spheres(outwards and inwards propagation).
The value of the different variables vary from system to system.

P_t	Emitted energy from transmitter
G_t	Gain of transmitter(antenna performance indicator)
A_r	Aperture of receiver(antenna performance indicator)
σ	radar cross section of target
F	pattern propagation factor
$4 \cdot \pi \cdot R^2$	surface of a sphere, in the formula two spheres are modeled (outwards and inwards propagation)

The formula emphasizes that there is a correlation of how far the signal has traveled and the reflected energy.

2.1.1 Frequency Bands and Wavelengths

RADAR systems are designed to work in specific bands and only receive reflections of a predefined signal or signals - hence a RADAR system is highly specialised for a certain task and not a general system.
The bands are in certain herz ranges, and most of them have a letter denoting the band (secret WWII codenames), the letters are still used as they were declassified after the war. Table 2 has a overview of the different bands. Different bands excel at detecting different phenomena.
There is a correlation between frequency and wavelength. This can be expressed using the following equation (Eq. 2).

$$f = \frac{C}{\lambda} \text{ or when isolating wavelength, } \lambda = \frac{C}{f} \tag{2}$$

On figure 2 i have added some calculations to illustrate the wavelengths for the bands. The higher the frequency the smaller the wavelength. The size of the wavelength directly correlates with resolution and how well objects of certain dimensions are detected.

2.1.2 Doppler Shift and RADAR

The diagram shown on figure 2 depicts a simple pulsed doppler RADAR system. It works by using a reference signal and comparing it to the received signal. This way it is able to determine the doppler shift using phase comparison.

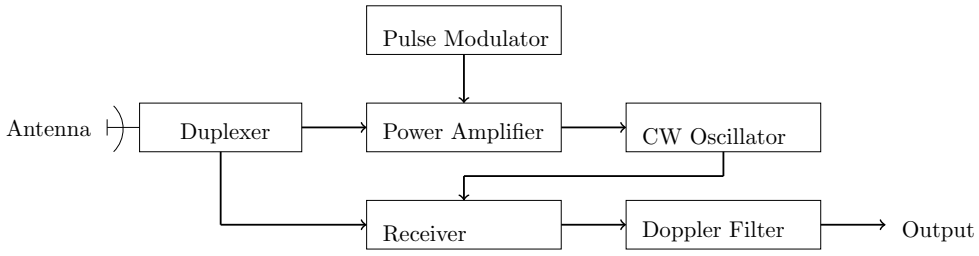


Figure 2: Block diagram of pulsed doppler radar

The doppler shift is used to determine whether a target is in motion or not. The best way to describe the doppler shift is as change of frequency for given

Band Designation	**Nominal Frequency Range**	**Specific Frequency Ranges for RADAR based on ITU Assignments in Region 2**	**Wavelengths**
HF	3-30 MHz		100 - 10 m
VHF	30-300 MHz	138-144 MHz 216-225 MHz	10 - 1 m
UHF	300-1000 MHz	420-450 MHz 850-942 MHz	1 - 0.3 m
L	1-2 GHz	1215-1400 MHz	0.3 - 0.15 m
S	2-4 GHz	2300-2500 MHz 2700-3700 MHz	0.15 - 0.075 m
C	4-8 GHz	5250-5925 MHz	0.075 - 0.0375 m
X	8-12 GHz	8500-10680 MHz	0.0375 - 0.025 m
K_u	12-18 GHz	13.4-14.0 GHz 15.7-17.7 GHz	0.025 - 0.016667 m
K	18-27 GHz	24.05-24.25 GHz	0.016667 - 0.011111 m
K_a	27-40 GHz	33.4-26 GHz	0.011111 - 0.0075 m
V	40-75 GHz	76-81GHz 92-100 GHz	0.0075 - 0.004 m
mm	110-300 GHz	126-142 GHz 144-149 GHz 231-235 GHz 238-248 GHz	0.002727 - 0.001 m

Table 2: IEEE standard frequency letter-band nomenclature[12, p. 12]. Added some calculations to illustrate the size of waves in that range.

signal in motion - a frequently used example is a siren on a moving emergency vehicle. The sound changes as the vehicle passes by, while moving closer it sounds in one way, and while moving away it sounds differently.

2.1.3 Interim conclusion

The previous section presented how RADAR systems utilize physical phenomena to sense the environment relative to the receiver. This establishes the minimum requirements for what is needed to be able to produce the required data for the planned machine learning investigation in section 5.

It is apparent that to reproduce the previously presented phenomena one needs to have a system supporting the tasks of being able to transmit and receive

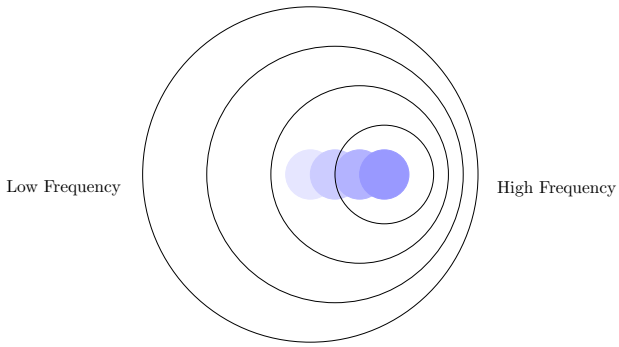


Figure 3: Example of Doppler shift on a moving object

signals.

While signal processing could be interesting to have in the system - it might be a bit more interesting to work with an almost raw reception of echoes than something already processed for the later experiments with machine learning.

Therefor the requirements for a system useful for producing data for the proposed ML experiments, is a system which is able to replicate RADAR at the signal level.

While the most obvious approach to reproducing signals from a RADAR system would be to acquire a RADAR system it is not the primary choice. Actual RADAR systems are quite expensive, specialized for certain tasks and require expertise to use.

Same can be said about simulated systems, however they are usually not as expensive as physical RADAR systems and it is usually easier to perform systematic tests in a simulated system.

The next sections will present the exploration of some of the candidates considered for RADAR simulation.

2.2 Candidate Environments

I have explored a select few environments to produce RADAR simulation data, this is far from an exhaustive list and during the course of the project i have discovered more alternatives, but due to time constraints these paths have not been explored and will be mentioned in section 7.

The candidates that i however have explored, range from known solutions such as matlab and octave, to less known suites such as open source Flexible Extensible RADAR Simulator(FERS).

Matlab is a highly favored environment for RADAR simulation when bundled with RADAR specific plugins.

The internet has many tutorials of how to model RADAR domain specific situations using the Phased Array Toolkit.

The downside is that Matlab is not free software and neither are all the plugins such as the phased array toolbox.

A commonly used alternative to matlab is a toolbox named Octave. Octave is like matlab also a toolkit aimed at doing mathematical models and calculations. Octave supports most Matlab code as it was designed with support in mind [6].

However it is difficult to find plugins for simulating RADAR systems but not for digital signal processing, which makes Octave a suitable candidate for DSP but not for simulating RADAR.

While the research i did does not necessarily rule out the existence of plugins with RADAR specific purposes the lack of availability on the main plugin sites such as octave-forge[1] support the claim that it is not as common as in matlab.

Another tool which has been explored is a result of a phd. thesis done in 2008,it is a software suite called *Flexible Extensible RADAR Simulatior*[2], FERS, which is released under the GNU GPL 2.0.

First and foremost FERS might not be the most modern simulator out there but it is free to use.

[1] `https://octave.sourceforge.io/`

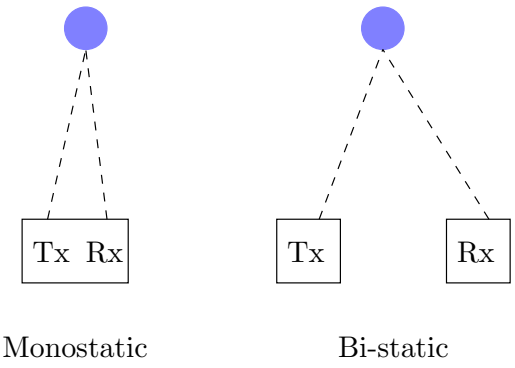


Figure 4: Image of different radar configurations. Monostatic configuration has transmitter and receiver at one location, Bi-static has them placed at different locations.

The FERS suite is able to simulate different types of RADAR systems such as mono or bi- static systems as seen in figure 4.

The strength of this simulator is the flexibility, which was one of the main design goals in the fers thesis[1, p. 5].

The possibility of being able to place different objects in three dimensional space and decide whether to make them static or dynamic opens up for rather sophisticated simulations.

The following sections will contain my experiences with some of the considered software suites using a simple experiment which is presented below. Before the experiment is described some formulas describing limiting factors of systems will be presented.

2.3 RADAR Performance Metrics

A RADAR systems performance relies on the different choices in configuration, choices such as frequencies, pulse duration, or listening window. Other more complex systems also apply multiple antennae which makes it even more tricky to determine performance metrics.

The system setup which will be used in this study will be kept as simple as possible with only one antenna.

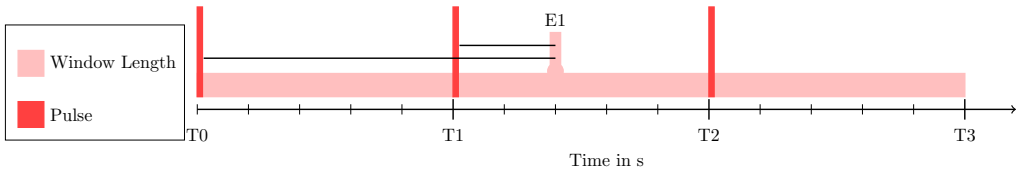


Figure 5: Example of Range ambiguity

Some formulas exist for expressing performance characteristics of systems, such as RADAR maximum range or maximum unambiguous range.

Unambiguous range RADAR is about ranging, hence a performance metric that expresses how far the system is able to reliably predict the distance an echo originates from is quite useful.

The smaller windows or pulse period(T_p), the higher the risk of second time around echoes or echoes that have spent longer time in transit than T_p.

This means that echoes that spend longer time in transit than the time in between pulses have ambiguities as seen on figure 5, where there is an uncertainity if it is caused by T_0 or T_1.

$$T_p = \frac{1}{prf} \tag{3}$$

$$R_{un} = \frac{c * T_p}{2} \tag{4}$$

$$R_{un} = \frac{c}{2 * prf} \tag{5}$$

Maximum Range Another useful metric is an indication of how remote targets the system can detect.

$$R_{max} = \left[\frac{P_t \cdot G \cdot A_e \cdot \sigma}{(4 \cdot \pi)^2 \cdot S_{min}} \right]^{\frac{1}{4}} \tag{6}$$

The value S_{min} is the minimum detectable signal, this is specified in the receiver as P_r.

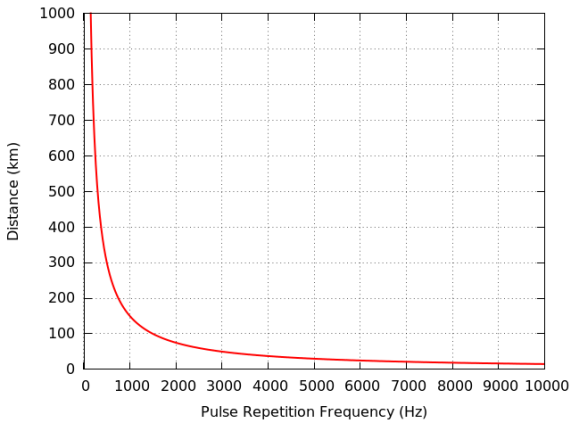


Figure 6: Image depicting the relationship between PRF and unambigious distances

2.4 Experiment

I have fabricated a small example which is a simple scenario with one target placed a distance away from the radar site. The radarsite is modelled as a pulsed monostatic pencil beam system.

The system parameters have been set to be similar to a mid-range air-surveillance radar as mentioned in Skolnik[12, p. 3] where a pulse repetition frequency of 1000Hz is used. This means that the range at which it is possible to reliably determine distance of echoes, max unambiguous range R_{un}, is roughly around 150km or more formally applying equation 5.

Looking at figure 6 it is possible to get an intuition of how much the PRF can influence the max unambiguous range.

The radar equation(eq. 1) was used to calculate the values returned from the target and compared to the results from the simulation.

Given a target at a position which is placed a distance d away from a radar-site, the returned power, P_r, is equal to the result from equation 1.

The values for R_t and R_r can be substituted by d, seeing a monostatic system is being modelled, which means that both the receiver and the transmitter are co-located.

Construct	Contents
Simulation Params:	start and end time(Seconds), simulator rate(Hz), export formats(csv, fersxml, binary)
Pulse:	power(Watts), carrier(Hz)
Clock:	Frequency(Hz)
Antenna:	dependant on antenna type, diameter or alfa,beta and gamma values
Platforms:	location, rotation, movement

Table 3: Table depicting the high-level structure of a fersxml file

The transmitter gain is approximated as if the target is in the mainlobe of the antenna radiation pattern. The aperture is defined by wavelength $A = \frac{\lambda^2}{4\pi}$. Wavelength is defined using frequency and speed of light as seen in eq. 2. The propagation factor, F is approximated as being lossless, hence the value 1 is used. The results can be seen on table 4, however only one simulation environment has been compared to the calculated values.

Matlab

Matlab is as mentioned earlier a commercial software suite. It has a broad user base, ranging from academia to commercial use. The main obstacle has been getting access to the RADAR toolkit. While it is possible to obtain 30-day trials to the core matlab suite and opt-in for plugin trials this does not seem like a solid approach. Either one ought to commit to the platform and invest in the suite and plugins or find other alternatives for doing simulations.

The subject of whether changing to a matlab simulation environment is preferable or not can be explored in the future.

FERS

The FERS suite is free software which is highly favored. The suite uses custom XML files following a scheme where one defines all the different parameters as seen on table 3.

The XML inspired format is however very bloated compared to the more

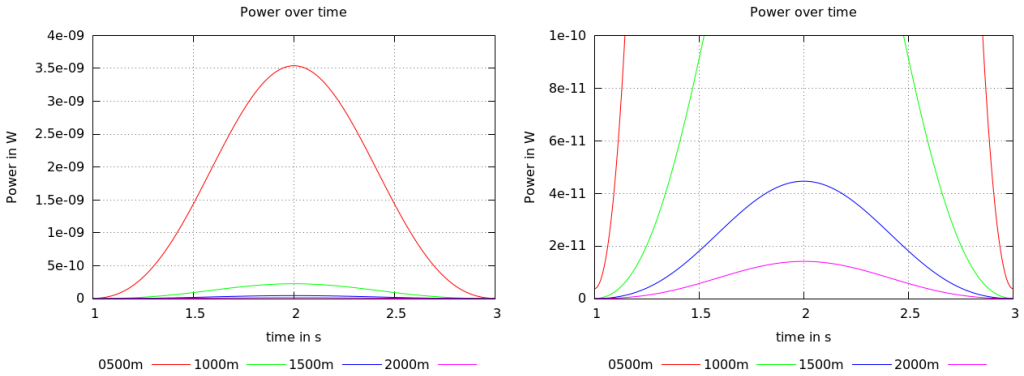


Figure 7: Distance test using FERS, 500m, 1000m, 1500m and 2000m targets. Same time window, different y-scales to illustrate the loss

popular and lightweight javascript object notation(json) format which is preferred at the time of writing.

Simulations are defined using the aforementioned .fersxml files to set up simulation parameters.

2.4.1 FERS configuration

As mentioned earlier a monostatic system with a pencil beam was to be mimicked. This requires an antenna with high gain in one direction, and lower gain in the remaining directions.

Using the built-in pattern functions in the FERS simulator the radiation patterns have been calculated to decide which one suits the simulation criteria the best. One is a sine cardinal(sinc) function pattern and another is a parabolic pattern. The sinc pattern is calculated based on equation 7 and can be seen on figure 8. The parabolic pattern is calculated by the equations 8 to 10 and is depicted on figure 8.

At first glance it seems as the sinc pattern fits the model the best. However if one increases the diameter of the parabolic dish, the gain in the mainlobe[2] increases very rapidly.

[2]Mainlobe is the symmetric structure centered around theta 0 of the pattern function

Distance	Calculated (Eq. 1)	Simulated(FERS)
500m	1.8116e-5	3.540141e-09
1000m	7.4288e-7	9.149915e-11
1500m	2.2365e-7	1.812399e-11
2000m	7.0767e-8	5.740083e-12

Table 4: Table with calculated values compared to simulated

The pattern rendered is calculated with a dish of size 3, changing the size to 10 increases the gain in the mainlobe by a factor of 10.

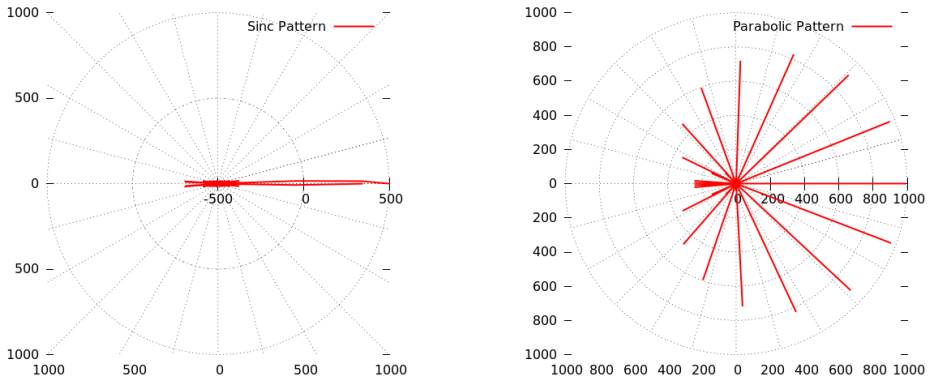


Figure 8: Left: Sinc Antenna Pattern, Right: Parabolic Antenna Pattern

$$G(\theta) = \alpha \left(\frac{sin\ (\beta\theta)}{\beta\theta} \right)^{\gamma} \tag{7}$$

$$G_e = \left(\frac{\pi d}{\lambda} \right)^2 \tag{8}$$

$$x = \left(\frac{\pi d}{\lambda} \right) sin\ \theta \tag{9}$$

$$G(\theta) = G_e \left(\frac{2J_1(x)}{x} \right)^2 \tag{10}$$

2.5 Interim Conclusion

The access restrictions to MATLAB made it less desireable for my project and i gravitated towards open-source alternatives.

The choice between an octave based simulation compared to a FERS based one was made on the fact that i was not successful in finding any RADAR specific octave plugins, and therefore settled with FERS.

It is apparent that if one had the funds to invest in a professional suite such as Matlab with the different RADAR toolboxes this can be worth considering in the future.

The simple experiment which was described, with a target placed at different intervals from the RADAR system, was done in FERS. The values are off as can be seen on table 4, however looking at the exponents of the values they have similar order of growth. This might be some approximation or factor that has not been included in the calculations. While the values do not match entirely the FERS simulator is used for further studies as the cause is most likely an oversight in the calculations done.

2.6 Generalizing Data Generation

What really makes simulation environments powerful, is the possibility of using scripts to automate work-flows. Especially bash and python are powerful tools for this.

The FERS suite expects XML files with loads of information about a simulation setup. When doing variations of such simulations it seems unnecessary to be forced to write entire XML files for each experiment when only a subset of the file needs variation.

For doing general tests it made sense to have a standard setup for the simulation environment, and inject different objects as needed. This resulted in me having a '_general' folder in every test folder. This folder contains general simulation parameters that describe parameters such as the antenna,radar site, and simulation duration.

Instead of having big verbose XML files something named .meta files have been presented. In short the meta file is a XML file with references to entities,

TAG	Description
<dxp>	delta x in positive direction
<dxn>	delta x in negative direction
<dyp>	delta y in positive direction
<dyp>	delta y in negative direction

Table 5: Table of tags for sed replacement

namely the entities which are externalized to the '_general' folder.

The general structure for a .meta file is as follows:

- 3 lines for bash scripts containing number of tests to generate, and variations of x and y values.
- Includes of external entities from '_general' folder
- Targets in the simulation. With special position tags these can also have varying positions.

This means one is only required to change simulation parameters in the externalized entities rather than every different subfolder when conducting experiments.

Native XML supports tags for including external entities, <xml:include>, combined with a XML parser tool the majority of the substitution can be done. The XML parser used in this case is *Xmllint*[16], which validates intermediary files and saves .fersxml files. The intermediary files are .meta files in which substitutions have been made ie. coordinate varied by a predefined factor per test variation. The substitustions are handled by Stream Editor(*sed*)[13]. The reason for having .intermediary files over doing direct modification in .meta files is to preserve the modularity in between test executions. If the meta was altered the special tags would be changed to ordinary tags, and during the next execution this would have no effect other than running the test.

The script is written such that *sed* does the substitutions and saves it to the intermediary files and *xmllint* does the xml validation on intermediary files and saves output as .fersxml files.

2.7 Scalability of FERS Environment

After selecting a simulator environment additional tests where performed to investigate how well it handled multiple targets. This meant that the previous approach of only having few hardcoded targets need to be revised. Another variation to the generalization was introduced to overcome the nuisance of having to produce multiple targets in bulk.

The previous approach of using xml includes would require just as many includes and references to files as targets. Hence using xml include seemed like overkill, which led to the solution that instead of letting *xmllint* handle the substitution *sed* was used. The xml include requires entities to be distinct files - hence for a certain amount of targets one needs the same amount of distinct files, aswell as the same amount of include references in the meta file. The reasoning for applying *sed* is the fact that rather than using an arbitrary amount of include tags all referencing to distinct files, it is possible to introduce a tag which *sed* searches for and replaces with the contents of just one file.

The downside is the lack of validation during the replacement step though, however this is done in the later step where one converts the meta file to an fersxml file. This results in errors getting caught, alas a little to late. This is avoided by not doing ill formed xml files other than having more than one entity seperated by an empty line in the replacement file.

The tag which has been introduced is <**TARGETS**>, and the script uses a file called platforms.xml as the replacement.

It was discovered that when introducing additional targets the number of responses varied as seen on figure 9.

The number of responses are directly proportional to the amount of objects in the simulation environment. Furthermore it has been experimentally proven that placement of objects has an effect on the simulator results. This can be seen on the graph in figure 9. One scenario is where all objects are placed deterministically(inline) and another where they are randomly placed. The expectation was that blocking line of sight would keep the echoes constant despite the number of targets introduced. This is however not the case as responses are increasing linearly. The FERS paper also states that the simulation is done on receiver-target pairs[1, p. 91] which explains why all

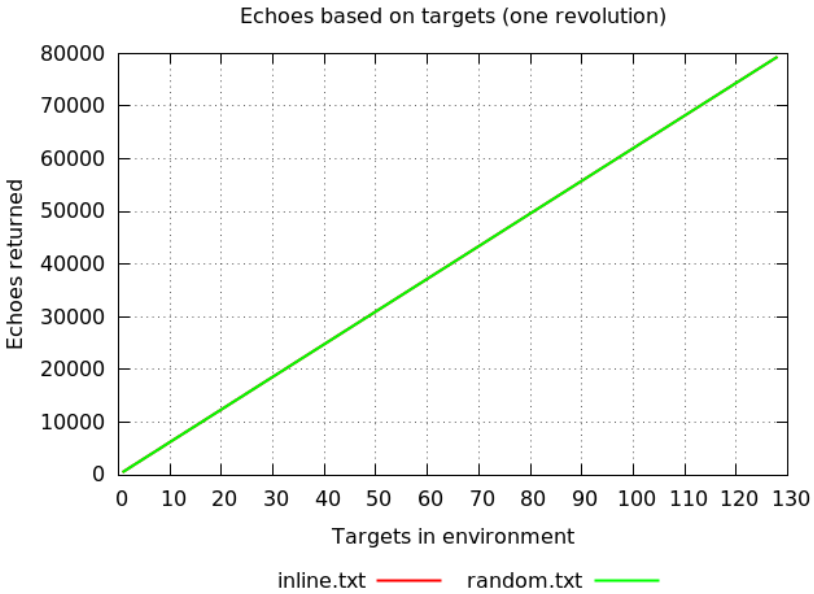


Figure 9: Testing the simulator

echoes are added despite the blocking line of sight.

2.8 Quantifying Experiments

The generalization of experiments using the FERS simulator is one step on the way to quantify the experiments. With the introduction of .meta files one can change one line to produce an arbitrary number of tests.
And by writing the markup in a certain way one can vary the location of objects in every simulation.

2.8.1 The Toolchain

Once a simulation suite has been defined with general simulation parameters and different test variations the folder structure is similar to what is shown below.

_general/	dir
east/	dir
west/	dir
runall.sh	script

Running the entry script 'runall.sh' traverses every sub-directory as seen on listing 1.

Listing 1: code/runall.sh

```
1  #!/bin/bash
2
3  for dir in `find . -maxdepth 1 -mindepth 1 -type d`; do
4    if [ "$dir" != "./_general" ]
5    then
6    cd $dir
7    sh ../../../util/generate_tests.sh
8    find -name "test*.fersxml" -exec ../../../util/runtest.sh '{}' \;
9    cd ..
10   fi
11 done
```

In each sub-directory it executes the script 'generate_tests.sh' which can be seen on listing 2. The generation script builds the required .fersxml files from the meta information, followed by a lookup of every .fersxml file present in the directory post generation.

Every .fersxml file is then processed using the 'runtest.sh' script as seen on listing 3.

Listing 3: code/runtest.sh

```
#!/bin/bash

# Get the basename of xml file, omit extension
base=`basename $1 .fersxml`
mkdir -p $base
# Get parent directory name
par=${PWD##*/}
cd $base
# Execute sim, redirect stdout and stderr to logfile
../../../../bin/fers ../$base.fersxml > $base.log 2>&1
echo 'Simulation performed '$base' in '$par
```

Once the simulation data has been generated the folder structure complicates the task of preparing data for ML slightly, due to them being spread across many folders and having the same name.

The extraction of files has been automated using the 'extractall.sh' script.

Listing 4: code/extractall.sh

```
#!/bin/bash

# Targetfolder - create log file aswell
base='sampledata'
mkdir -p $base
touch $base/test.txt
for dir in `find . -maxdepth 1 -mindepth 1 -type d`; do
  # Ignore the new directory and the general settings
  if [ "$dir" != "./_general" ] && [ "$dir" != "./$base" ]
  then
  # Treat every testcategory as a label
  label=`basename $dir`
  echo $label
  # Prepare target folder
  mkdir -p $base/$label
  cd $dir
  # Lookup and logging
  data=`ls */*mono1.csv`
  echo $dir' \n\n' >> ../$base/test.txt
  echo $data >> ../$base/test.txt
  c=1
  # Copy all data files and rename them
  for d in $data; do
    cp -- "$d" ../$base/$label/$c.csv
    c=$((c+1))
  done
  cd ..
  fi
done
```

This results in all files being copied to one folder with unique names. Which makes it simpler to process the data later as will be presented in section 4.

Listing 2: code/generate_tests.sh

```
#!/bin/bash
if test -e "test.meta";
then
  # Extract variation params (# of tests, delta x, delta y)
  in=$(head -1 test.meta)
  in1=$(head -2 test.meta)
  in2=$(head -3 test.meta)
  tests=${in##*=}
  dx=${in1##*=}
  dy=${in2##*=}

  # Loop number of tests
  for n in $(seq $tests)
  do
    # Get X and Y for this test
    nx=$((dx*n))
    ny=$((dy*n))
    # Replace target tags with targets.xml if present and generate intermediary
    if test -e "targets.xml";
      then
        val=`cat targets.xml`
        sed  '/<TARGETS>/ {r targets.xml
        d} ' test.meta > test.intermediary
      else
        cat test.meta > test.intermediary
    fi
    # Ommit the first 3 lines of the intermediary and validate fersxml
    # do sed replacement on dynamic targets
    tail -n +4 test.intermediary | xmllint --xinclude --output test${n}.fersxml --noent -
    sed -i 's/<dxp>0.0<\/dxp>/<x>'"$nx"'<\/x>/' test${n}.fersxml
    sed -i 's/<dxn>0.0<\/dxn>/<x>'"-$nx"'<\/x>/' test${n}.fersxml
    sed -i 's/<dyp>0.0<\/dyp>/<y>'"$ny"'<\/y>/' test${n}.fersxml
    sed -i 's/<dyn>0.0<\/dyn>/<y>'"-$ny"'<\/y>/' test${n}.fersxml

  done
  echo 'FERSXML files generated, running tests'
fi
```

2.9 Interim Conclusion

The last few sections have presented some extension to the FERS simulation suite, such that defining experiments with repeated patterns is simpler.

This has been achieved by using bash scripts and different programs such as sed and xmllint.

The above does ease the workflow, as one can define a suite of tests by one general folder with simulation parameters and additional folders containing the actual test setups. Each setup can be defined with variation parameters, such as how many different tests to perform, and size of the constants to apply to XY coordinates on targets.

3 Interpreting Signals

The field of signal processing is a natural extension to RADAR systems. While one could use RADAR exclusively as a 'dumb' sensor, the different techniques presented in this section lay the foundation for transcending from something simple to something remarkable.

Interpreting the response trains of RADAR systems is no trivial task, and as such one needs tools to aid the system in making sense of the responses, such as filters and statistical analysis. Much like in machine learning, which also builds on concepts from statistics, it is crucial to remove as many uncertainties as possible to improve precision of predictions.

This section will present some standard tasks that digital signal processing(DSP) is used to solve, followed by some standard ways of processing signals to remedy this situation of uncertainties.

While doing the exploration it became apparent that the scope of the project might be a bit too extensive, hence this part might seem a bit shallow. It has been included such that one can have an alternative angle to the techniques used in section 4 for data preparation before ML.

3.1 Common Tasks Solved Using DSP

Now that the basic operations of RADAR have been presented, it is obvious that these systems have potential to solve some complex tasks. Some of the simpler tasks that RADAR systems have been used to solve are detection and ranging. The more complex ones are moving target indicators (MTI), tracking, and target identification.

Detection Identifying targets within range of the system. Hence one of the biggest obstacles is to efficiently filter out clutter. Clutter might make it difficult to distinguish a correct response compared to a undesired one.

Ranging One of the most common tasks that RADAR is used for, it is about sensing the distance to the target. This is done post-detection as .

The formula describing range to a target from a system is expressed as

$$R = \frac{t \cdot c}{2} \tag{11}$$

, where R is the range, c is the speed of light and t being the roundtrip time. The above formula assumes that the receiver and transmitter are colocated, hence the division with 2.

Moving Target Indication The change in frequency which is named the doppler shif is a used to indicate whether a target is moving. As the frequency is affected by whether something is closing in or distancing itself from the RADAR. These phenomena can be expressed using either negative or positive values.

Tracking Tracking requires the system to be able to distinguish different targets from one another. Additionally it requires the system to be able to correlate the same target to a path over the course of multiple scans.

Target Identification is about determining what is moving and not where it is. This is done by having sufficient resolution by minimizing the beamwidth of the antennas mainlobe, such that the beamwidth is smaller than the target. This results in different returns and this can make an approximation of the shape of the target.

3.2 Processing

For each mentioned task different techniques are utilized to minimize the uncertainties present in the responses. This can be using data from multiple pulses or applying filters to data. I will briefly present some techniques, just to hopefully present an idea of what has been done earlier to remove uncertainties in data.

It is done using SciPy[11], which has a signal processing library.

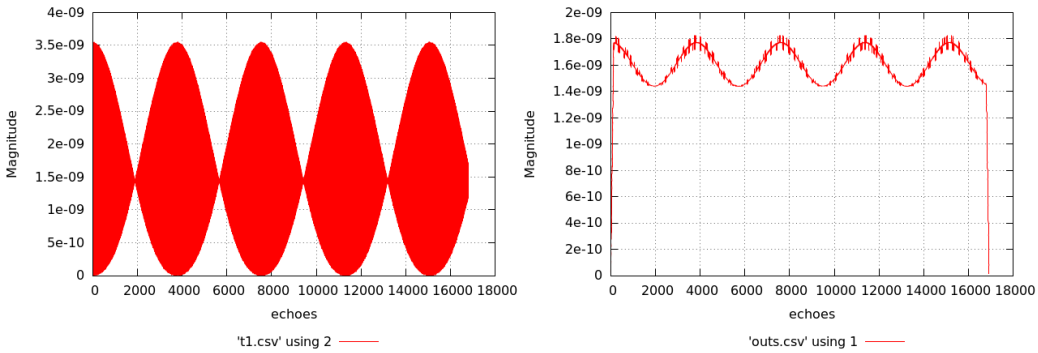


Figure 10: A signal with 2 targets present respectively at east and west. Left is raw, right is post correlation.

3.2.1 Integration of pulses

Integration in signal processing is the summation of pulses. Depending on the frequency of the system the beam might dwell on the same target for multiple pulses. This results in multiple responses originating from the same target and hence this indicates a higher chance of a target being present.

3.2.2 Filtering

Applying filters that improve certain things in the signal is great for solving the above mentioned tasks. I have tried using scipy on a signal with 2 targets. This is a minor investigation to see the effects of applying different filters.

Match Filtering

This technique is about finding peaks. It is a variation of a correlation where one tries matching a specific pattern in the input signal. The key being trying to match a known signal. The trick is to 'mirror' the received signal and then add both together[12, p.278,281].

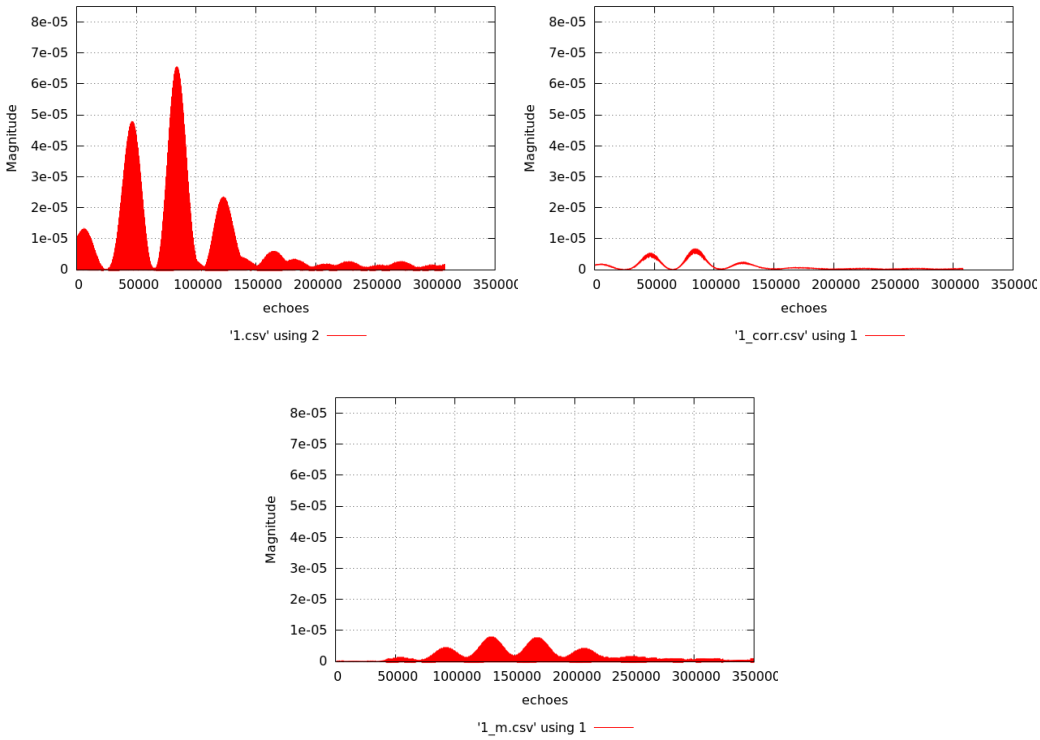


Figure 11: A signal with 1 target east and 5 random clutter samples. Left is raw, right is post correlation, bottom is post matching.

Correlation

As can be seen on figure 11 applying filters to data can have a big effect on the signal. Especially clearing up peaks such that it is easier to find the patterns one is searching for.

4 Data Preparation

Applying machine learning to any domain requires preparation of data. Especially when doing supervised variations, it makes sense to guide the data in some uniform direction. This can be windowing the data in certain block sizes or converting non-scalar values to scalar values. Additionally it it is crucial to fill in gaps in data sets - such as missing values or 0 values if the algorithms are not expecting data of that sort.

The data which has been produced so far consist of tuples of numeric values.

$$(timestamp, Power_{returned}, Phase, Doppler) \tag{12}$$

All of the above values are represented using double precision numbers.

4.1 Sampling

The RADAR data is a sequence of echoes stored in CSV format. In the simulations one scan is synonymous to one revolution. The duration of one scan cycle is kept constant across all test variations, as are the majority of the system parameters.

Every sample should conform to one golden rule which is *samples should be of equal length*. The only variations introduced to the samples are targets and clutter. These variations are however enough to complicate the preparation of data, since having multiple targets and clutter results in varying amounts of echoes returned(As shown in the tests on figure 9 in section 2.7).

The field of data science has lots of literature on how to prepare different types of data for ML. One of the factors that can influence data sets negatively are missing values, in the current setup with simulated data it is not likely to happen unless modeled.

This issue has been handled by working with blocks of data each corresponding to one angle of degrees or a fraction, which means that anything with a time stamp within a window that covers 1 degree of a revolution is processed and merged into one value.

In the end this means that 360 values are present per scan cycle. This results in all samples, which are also scan cycles, being of equal length of 360 data

points, which is due to the restraint that samples should be of equal length. This is one way to prepare data for the later investigation described in section 5.

Manipulating csv in python is done easily using external libraries, such as Pandas[7]. Pandas excels at csv manipulation, as one can access data row by row, take subsets of the data using index numbers, filter out columns and other useful tasks.

At the time of writing only one approach has been applied to the data, which is taking a predefined number of samples per scan, one per degree. This approach is also known as binning [5, p. 64]. It also introduces errors when replacing values with means, as it is sensitive to the number of samples in the bin and outliers in the bin [5, p. 88].

$$T_{unit} = \frac{scan\_time}{360} \tag{13}$$

The echoes E in one time unit of size, T_{unit}, are then processed by calculating the mean of every row in a column and merging into one value. This will result in every sample being of 360 values corresponding to the mean values for that angle chunk. If one compares the effect of binning to the previously presented correlation, one can see that pattern wise they show similar graphical representations.

A python executable has been written which takes a directory of raw test simulation data and prepares it into smaller blocks of scan samples, such that the data conforms to the guidelines for the ML library which has been chosen for the exploration of ML in section 5. An overview of the newest version of the sampling script follows:

permute.py	
void main(args):	arg1: target dir arg2: source dir
Sequence(Sequence) extractscan(src, scan_time, prf)	src : filepath, scan_time : seconds, prf : frequency
(float,float,float,float) anglesample(df)	df : A dataframe containing multiple rows

This script ensures that every sample taken from each simulation is of equal length.

Currently the script has scan time and prf as constants in the main function, which is deemed bad practice since it makes the script less flexible if one wants to alter the radar system configuration one is forced to alter the source code. A better approach would be to elevate the values as being arguments for the main function to improve flexibility. The flow in the script is as simple as traverse all files in the source folder, apply the extractscan function to a file, which applies the sampling function and returns a list of samples from that file.

Listing 5: code/naive_permute.py

```
11 def processample(dataframe, n):
12     extract = 0
13     df = pd.DataFrame(columns=('Time','Power','Phase','Doppler'))
14     sample = dataframe[extract*n:extract*n + n]
15     while(extract*n < size(dataframe)):
16         #update value
17         a = sample[0].mean(), sample[1].mean(), sample[2].mean(), sample[3].
               mean()
18         df.loc[extract] = [ a[i] for i in range(4)]
19         #new sample
20         extract = extract + 1
21         sample = dataframe[extract*n:extract*n + n]
22     return df
23
24
25 # Extract values for scan cycle
26 def extractScan(source, scan_time):
27     print('␣␣␣Extracting␣file␣%s' % source)
```

```
28      list_of_scans = []
29      angle_time = (scan_time/360)
30      try:
31          data = pd.read_csv(source, header=None)
32          # determine echo number
33          t = 0
34          first = data.iloc[0][0]
35          max = size(data)
36          n = 1
37          scan_t = scan_time
38          print('␣␣␣Data␣properties␣sz␣%i␣t␣%f' % (max ,first) )
39          while (n < max and scan_t > t ):
40              t = data.iloc[n][0]
41              n=n+2
42          #sample data
43          scan_size = n
44          print('␣␣␣Scan␣is␣%i␣rows' % (scan_size) )
45          samples_taken = 0
46          while (samples_taken * scan_size + scan_size) < size(data):
47              low = samples_taken*scan_size
48              sample = data[low:low+scan_size]
49              list_of_scans.append( procesSample(sample, scan_size/
                    revolution_samples ) )
50              samples_taken = samples_taken + 1
51              print('␣␣␣Sampled␣%i␣scans' % samples_taken)
52          print('==␣File␣%s␣processed' % source)
53
54      except IOError:
55          print('No␣file␣named␣'.join(source))
56          sys.exit()
57
58      return list_of_scans
```

4.1.1 Training samples

The training has been done using both uncluttered data and cluttered data. The uncluttered data has a target placed at either north, south, west, or east from the RADAR system with a variable range. This has been used to automatically generate 500 test examples per direction, hence 2000 examples in total.

For static targets in uncluttered environments it makes no sense to sample more than 1 sample per file as this is just duplication as seen on figure 12.

However all the static samples have still been used despite the excessive amount of duplication.

The same is less likely to happen when using cluttered data, as seen on figure 13. The cause is due to the fers toolbox being bundled with a clutter

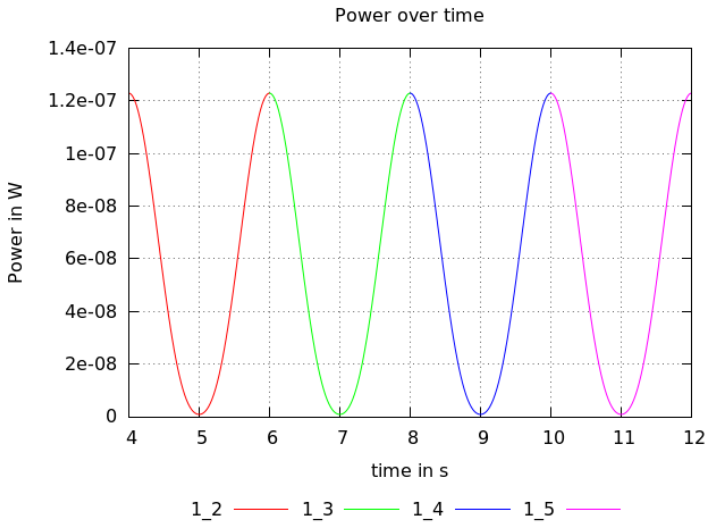


Figure 12: Example of consecutive samples in non-cluttered static environment

generation tool which is able to generate clutter at random coordinates and at random times.

Another observation which has been made is that as the distance between object and system increases the response power converges to zero. This is caused by targets being placed further away than the maximum range(as presented in section 2.3).

During the processing of data it was discovered that placing objects further than R_{max} makes no sense, so there is a limit on how many variations should be done when the delta values are placing objects further than the R_{max}

The 2000 test examples are roughly 3Gb of data stored in CSV format.

4.1.2 Testing samples

The test data is prepared like the training data. It is very important that one does not use any of the training data for testing, as this might give a wrong representation of how well the ML techniques perform.

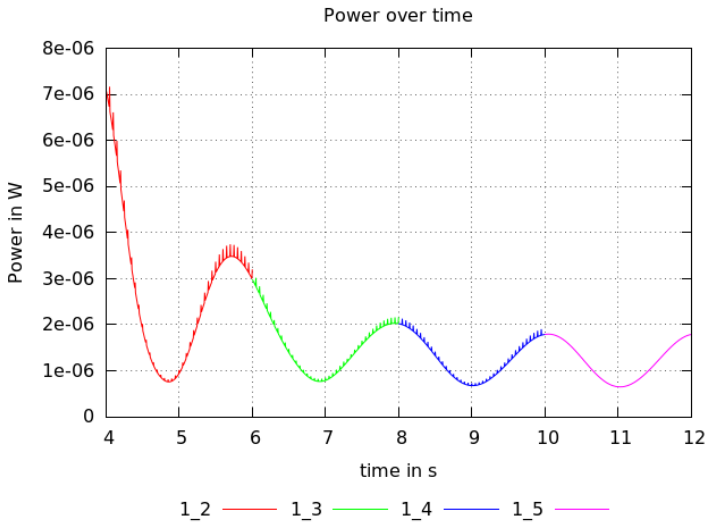


Figure 13: Example of consecutive samples in cluttered static environment

4.1.3 Scripting for Scalability

The naive data sampling script performs quite well on limited data sets. However it has not been written with scalability in mind. The need for a faster processing was discovered when doing the sampling on larger datasets. While there has been no official timing the naive version of the script ran for more than 25 hours. With roughly 1.5MB data per file, and 500 files per category this scales up to roughly 3 Gb of data ($1.5 * 500 * 4 = 3000MB$).

This led to a superficial investigation of the code, which resulted in some weaknesses being identified such as *Data being traversed in such a manner that every label directory is processed and sampled before writing anything to disc* and *that no parallelism has been applied.*

The culprits are linked, since sample data is not written while traversing the files, which retains data in memory longer and the other is not exploiting the fact that the processing of every file can be perceived as one single task which is highly scalable on modern day CPUs. To remedy the negative effects of having the workflow as seen in procedure 1, one should instead redefine the flow to what is seen in procedure 2.

Procedure 1: Naive Workflow

$T = \emptyset$
foreach $dir \in D_{src}$ **do**
  $S = \emptyset$
  foreach $f \in dir$ **do**
    $S_f \leftarrow$ list of samples $\{s_{f0}..s_{fn}\}$
    $S \leftarrow S \cup S_f$
  end
  $T \leftarrow T \cup (dir, S)$
end
foreach *(dir,S)* $\in T$ **do**
  foreach $s \in S$ **do**
    write to target directory D_{tar} and sub directory dir
  end
end

This restructures the flow such that every file is handled atomically, hence read file, sample file, and write samples to disc.

Procedure 2: Improved Workflow

foreach $dir \in D_{src}$ **do**
  foreach $f \in dir$ **do**
    $S_f \leftarrow$ list of samples $\{s_{f0}..s_{fn}\}$
    foreach $s \in S_f$ **do**
      write to target directory D_{tar} and sub directory dir
    end
  end
end

This makes it easier to implement a threadpool, and map the different files to tasks using a worker function.

Listing 6: code/permute.py

```
92      # if directory exists open sources, else open single file
93      if os.path.exists(sdir):
94          print('Source␣dir␣exists..')
95          source_folders = os.listdir(sdir)
96          print('Sources␣found..␣%s' % (source_folders))
97
98          for folder_name in source_folders:
99              #worker function
```

```
100             def worker(fname):
101                 base,_ = fname.split('.')
102                 es = extractscan( '%s%s' % (src_subdir, fname), scan_time,
                        prf )
103                 # Target Directory
104                 t_subdir = "%s/%s/" % (directory,folder_name)
105                 if not os.path.exists(t_subdir):
106                     try:
107                         os.makedirs(t_subdir)
108                         print('␣␣␣created␣subdir␣%s' % t_subdir)
109                     except Exception as e:
110                         print('Directory␣not␣created,␣something␣went␣wrong\n
                                %s'%str(e))
111                         sys.exit()
112                 # Write sample files
113                 smp = 0
114                 for sample in es:
115                     fn = "%s/%s/%s_%i.csv" % (t, folder_name, base, smp)
116                     sample.to_csv(fn, encoding='utf-8', index=None, header=
                            None)
117                     smp += 1
118                 print('␣␣␣Created␣%s␣samples␣from␣%s\n' % (smp, folder_name)
                        )
119
120             src_subdir = "%s/%s/" % ((sdir, folder_name))
121             print('\n+␣Traversing␣%s..' % src_subdir )
122             sources = os.listdir(src_subdir)
123             print('+␣Found␣%i␣files..' % len(sources) )
124
125             pool = ThreadPool(1) #bugs when using more than 1
126             print('pool␣created')
127             pool.map(worker, sources)
128             print('mapped␣workload')
129             pool.close()
130             pool.join()
```

The above code, which is an excerpt of the main method, is still buggy but it sped up the traversal of the 2000 files(3Gb) significantly from roughly 25 hours to just above 2 hours.

The script has been altered in three ways - one is *not retaining all the samples in memory longer than needed*, another is using the *pandas library filters on columns* rather than manually iterating over every row and checking if the time is lower than an upper limit, and the last alteration is introducing a *worker function and mapping files to tasks* such that threadpools can execute them in parallel.

Listing 7: code/permute.py

```
12 # dataframe: values from one degree of a scan
13 # return: one echo object
14 def anglesample(dataframe):
```

```
15      a = dataframe['Time'].mean(), dataframe['Power'].mean(), dataframe['
            Phase'].mean(), dataframe['Doppler'].mean()
16      return a
17
18
19  # Extract values for scan cycle
20  def extractscan(source, scan_time, prf):
21      print('␣␣␣Extracting␣file␣%s' % source)
22      list_of_scans = []
23      angle_time = (scan_time/360.0)
24      try:
25          data = pd.read_csv(source, header=None)
26          data.columns = ['Time','Power','Phase','Doppler']
27          max = size(data)
28          max_t = data.iloc[max-1][0]
29          print('␣␣␣Data␣properties␣sz␣%i␣MxTime␣%f' % (max ,max_t) )
30          #go through all datapoints (using timestamps)
31          s=0 #scan counter
32          hi=0
33          while hi < max_t:
34        #time bounds for scan
35              lo = (s*scan_time)
36              hi = lo + scan_time
37        # get rows in range
38              scan = data.query('%f␣<=␣Time␣<=␣%f' %(lo,hi))
39              a=0
40              df = pd.DataFrame(columns=('Time','Power','Phase','Doppler'))
41              while(a < 360):
42            # angle lower bound
43                  alo = (lo+(a*angle_time))
44                  ascan = scan.query('%f␣<=␣Time␣<=␣%f' % (alo,(alo+angle_time
                        )))
45                  df.loc[a] = anglesample(ascan)
46                  a=a+1
47              s=s+1
48              # if sample is uneven we are near the end and disgard
49              if size(df) < 359 :
50                          break
51              list_of_scans.append(df)
52          print('==␣File␣%s␣processed' % source)
53
54      except IOError:
55          print('No␣file␣named␣'.join(source))
56          sys.exit()
57
58      return list_of_scans
```

The threadpool implementation is however buggy, so instead it was done using 1 thread per pool and spawning 4 terminals running one python instance each processing one folder each. This was a quick way to achieve parallelism however it took some manual labor to copy the subdirectories into 4 different sample directories.

There are some suspicions as to where the problems with threading can arise,

but at the time of writing there has not been invested more time exploring the suspicions. The first suspicion is that it could be the windows subsystem for linux(WSL) having some compatibility issues due to it being a 'pseudo linux' distribution built with windows interoperability in mind and users should expect "many things to work and for some things to fail"[15], the other potential issue could be the python distribution(2.7.6) having some threading bugs.

On the other hand i ran the script using PyCharm which is installed on the windows system - which also ran into the same threading issues, which indicates it might be something entirely different.

4.2 Interim Conclusion

While the task of sampling from data-sets might be deemed a trivial task, one should not underestimate the importance of writing code with scalability in mind, since for supervised ML, there needs to be a significant amount of data to learn from. If the above is not the case, and the data is sparse, the predictions will be less precise, hence efficiently producing enough data in reasonable time is highly desirable.

This leads to the other observation that if possible one should try and use tested solutions as much as possible. This can be deduced from the speed improvement when traversing the same data-set using the naive permute script compared to the optimized script. Especially the utilization of the pandas library and its built-in column filters over manual look-ups of every element is likely to be the main contributor to the speedup.

4.2.1 Shortcomings

Some improvements could have been done by thoroughly investigating what was causing the performance issues, and how every improvement affected the overall processing time on a given workload over the course of multiple executions. The python script having bad performance was however discovered rather late in the process and hence not explored as thorough as is the norm.

Another shortcoming is the fact that the sample size is set to 360. This results in many data points being merged into single points. Increasing the

sample size to be a multiple of the prf and the scan time is most likely better for higher precision. In the generalized example of 1000Hz prf and 2 seconds revolution time this would mean echoes from 2000 pulses being merged to 2000 values or more precisely 1 value per pulse. The above alteration might be worth exploring in the future such that less information is lost when processing the data.

5 Machine Learning

As section 3 illustrated, it is possible to make some predictions about targets using rather simple techniques.

This introduces the question of why it is interesting to fit ML into the processing chain.

As more complexity is present in the response, complexity being multiple targets or clutter, more advanced filtering is required to distinguish the intended targets from the unwanted data.

This leads back to what was stated in the beginning, in section 1.1, which was that the random nature exhibited by sensory data from RADAR systems makes neural networks good candidates for processing data.

The data sets which will be used as a base for this ML study are produced using the toolchain presented in sec 2.8.1, and the data is split in 3 different sets, each being of a certain type.

The first set is a uncluttered set, the second has some clutter targets introduced, and the third has even more clutter targets introduced. The sets are common in the sense that the target of interest is placed at a predefined direction, ie. east, south, north or west.

After the first part with pure data sets, incremental sets have been produced. The above means that the sets have been merged such that two new sets are formed, one containing the uncluttered set and the less cluttered set, and the other containing all three sets.

5.1 Target Detection Using Max function

As has been mentioned in the previous section 3.1 target detection can be somewhat tricky to achieve in very cluttered environments, hence efficiently pre-processing data is crucial to achieve better detection rates.

In uncluttered data the peak is where the reflection is greatest, and most likely where a target is present. *This raises the question whether it would suffice to use a max function to make the decision.*

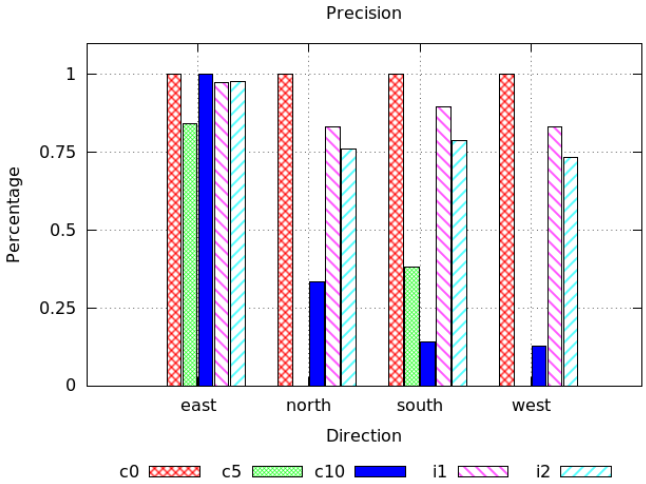


Figure 14: Max function predictions on training data, c0 to c10 are clutter sources, i1 is c0 and c5, i2 is c0,c5, and c10

The function in listing 8, implements a prediction function which finds the index of the maximal power value. The pandas library is utilized to find the index of the row with the highest P_r. Then by using the aforementioned index, the time stamp of the row is looked up and placed within one of the intervals for each direction. The rotation defined in the simulator rotates counterclockwise and starts to the right. This means a sequence of EAST, NORTH, WEST and SOUTH is repeated during one scan cycle.

The script has been expanded with a statistics function as seen on listing 9. The current version does not take recall into account as every sample falls within a proper category.

The result of applying the 'max_predict' function to the different samples can be seen on figure 14 and 15.

Additionally there is a graph depicting how many samples have been used for each test.

The data sets for cluttered data are somewhat smaller as can be seen on graphs 16 and 17.

The uncluttered set has approximately 14000 samples, the cluttered sets have 2800 each, totaling 5600 samples. There is an over representation of

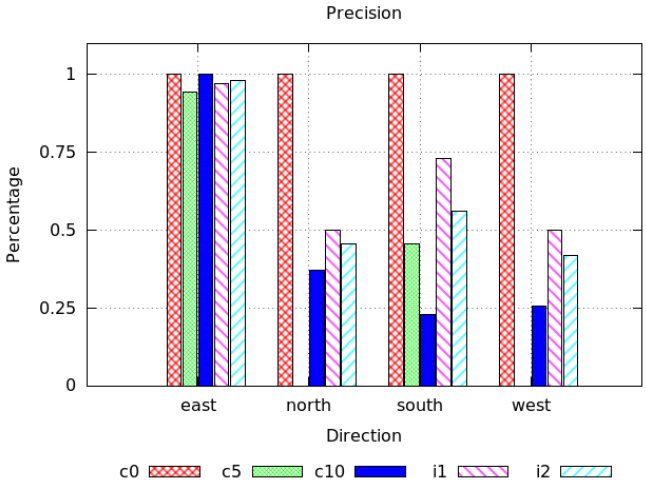


Figure 15: Max function predictions on test data, c0 to c10 are clutter sources, i1 is c0 and c5, i2 is c0,c5, and c10

uncluttered samples with a factor of 3. This is however only the case for the training sets. The test sets have been produced later an are as such all of equal size.

This has a impact during the training. In future studies it would be interesting to change the representation such that there is a different ratio between the variations.

5.1.1 Interim Conclusion

Once clutter is introduced to the sample data, the max function has declining performance. As can be deduced from the predictions, the randomness is clustered in the general direction of east.

While the peak of interest remains at the same angle or timestamp, the introduction of clutter increases the number of false predictions, due to clutter return power being higher than the return power from the desired responses.

This supports the claim, that detection in cluttered environments is non-trivial to achieve, and exploring the application of ML techniques to the domain is still of interest.

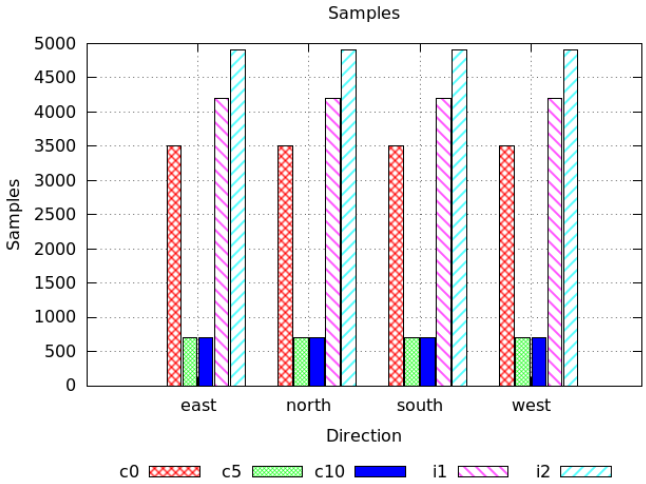


Figure 16: Max function samples on training data, c0 to c10 are clutter sources, i1 is c0 and c5, i2 is c0,c5, and c10

Shortcoming

The random clutter has not been generated properly. It has a gravity towards the eastern coordinate due to the variables used when generating the clutter all being of positive values. This means the clutter is placed somewhere within the first quadrant in the XY plane. With the RADAR being placed at zero coordinate it explains the gravitation of the predictions towards this direction.

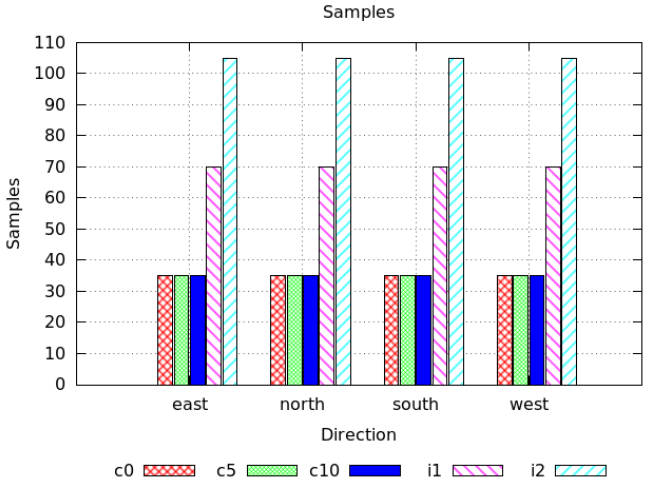


Figure 17: Max function samples on test data, c0 to c10 are clutter sources, i1 is c0 and c5, i2 is c0,c5, and c10

5.2 Target Detection Using Networks

The previously discovered error during configuration of simulations is however constant across all instances, and currently due to time restraints i am not to repeat all the data generation and sampling.

Using networks for doing predictions based on data series is not a novel idea, they have been succesful in solving tasks ranging from number recognition[4] to recognizing ekg[8] and other similar tasks.

The strengths of networks are their "high tolerance of noisy data as well as their ability to classify patterns on which they have not been trained"[5, p. 327] which fits well in this specific use case of training on cluttered or 'noisy' RADAR data.

While i had a brief glance at Google's tensorflow, which is built for Python, but however has APIs for other platforms such as Java, Go and C, i decided to go in a different direction. I ended up working with another popular framework also for python, named SciKit Learn(SKLEarn)[9]. The argument for selecting SKLearn is that i had some prior experience with it, which speeds up the entire process of going to prototyping.

5.2.1 MultiLayer Perceptron

A MultiLayer Perceptron is a neural network with multiple layers[5]. The network consists of one input layer with a predefined number of nodes matching the feature size, and one output layer which contains nodes matching the number of target classes.

The strength of MLPs is that they have the "[c]apability to learn non-linear models" and the "[c]apability to learn models in real-time (on-line learning) using partial_fit"[10].

Both strengths align with the nature of RADAR data as data can be both noisy and is usually processed in real-time.

The end goal is at some point being able to apply these networks to streams of data in real-time opposed to the current approach with already gathered data.

Network Topology Deciding a suitable network topology can potentially affect the prediction rates. The hidden layers are not fixed, they can vary both in number of layers and size per layer. There is no golden rule for which kind of network topology is most optimal, one is encouraged to use an experimental approach because "[n]etwork design is a trial-and-error process and may affect the accuracy of the resulting trained network."[5, p. 329].

5.2.2 Scaling Input Data

A common step in ML is to pre-process input, both training samples and test samples. This step can potentially improve the prediction rates of networks significantly.

Different types of data pre-processors, which are made readily available in the SKLearn toolkit, have been applied to the data. A brief presentation of the three will follow:

Standardisation This type of pre-processing prepares the data such that features are standardized by removing the mean and scaling to unit variance.

MinMaxScaling Scaling of data in a range, defaults to 0 and 1. This result in every feature being placed within the predefined range.

Normalizing In this variant the samples are scaled to a unit norm.

The different pre-processors have been tested with different networks as will be presented in section 5.2.3.

The general observation in this study is the fact that the normalizer seems to have the worst performance.

It scores roughly somewhere between 5-10% lower than the other data processors.

5.2.3 Optimizing Network Topology

The network has been optimized systematically. Multiple parameters can be tweaked to change how performant a network is for a specific task.

The MLP has been tested using a systematic approach that varies the hidden layer sizes. The literature states the optimal hidden layer sizes to be somewhere between the size of the output layer and the feature size.

Sizes greater than 100 have been attempted, however the strain which is put on the system during the tests and no noteworthy improvements are used as arguments as to why not pursuing that path, however it does make sense to explore it further in the future.

With 360 features having one connection to each neighboring hidden layer node the complexity of the network grows quite rapid once the hidden layers are scaled. This explains why the computations become increasingly slower when increasing the sizes dramatically.

Additionally the training sets, even when prepared, grow to a size of 300MB, which is traversed all the time when training different MLP classifiers. The 300MB cover just below 20.000 sample files.

The variable used for the constant size variations has been chosen arbitrarily as 32 nodes, and 3 samples have been taken with intervals of 32 values. The variable layer is between 24 and 96, with a variation of 12 nodes. With the above variations multiple network topologies have been constructed and tested on the different data sets, which will be presented in the following sections.

5.3 Multiple Data Sets

The baseline performance has been established using the max function. The max function has been run on multiple data sets, the more clutter is introduced, the worse precision the function exhibits. The current performance metrics are however biased due to clutter samples being produced in a specific area in the XY plane, as noted previously.

5.3.1 Uncluttered Data Sets

In uncluttered variations as seen on the different figures in appendix A the network topologies perform well overall. This was also expected, and as such not that interesting to spend too much time on.

There are some networks that appear less performant which indicate that these combinations should be avoided, but in the general case the tests on the uncluttered data sets indicate that for this particular problem with clean data, MLPs are just as good to use as the max function. The max_function is however not as computationally heavy as it does not need any training or anything before being able to estimate, but this is not that big an issues as once a set of well performing weights have been found they do not need to be updated.

5.3.2 Cluttered Data Sets

Performing the same experiments on cluttered environments with 5 randomized clutter sources yield different results as can be seen on figures 18, 19 and 20.

The overall performance is somewhat lower than in the uncluttered environment, however the decline in performance is nowhere near the drop experienced with the simple max function.

Another thing to note is however how the feature scaling influences results. The MinMaxScaler(fig. 18), which is scaling everything between 0 and 1 has the best performance overall with quite a few topologies still reaching about 1.0 precision. The Normalizing of data(fig. 19) on the other hand reduces the precision on many of the networks, reducing the precision to somewhere

in the range of 0.5 to at best around 0.8. The last variant of data scaling is standard scaling(fig. 20), it performs better than normalizing the data, but in general worse than the MinMax scaling. The majority of the networks with standard scaling stay around 0.7 precision.

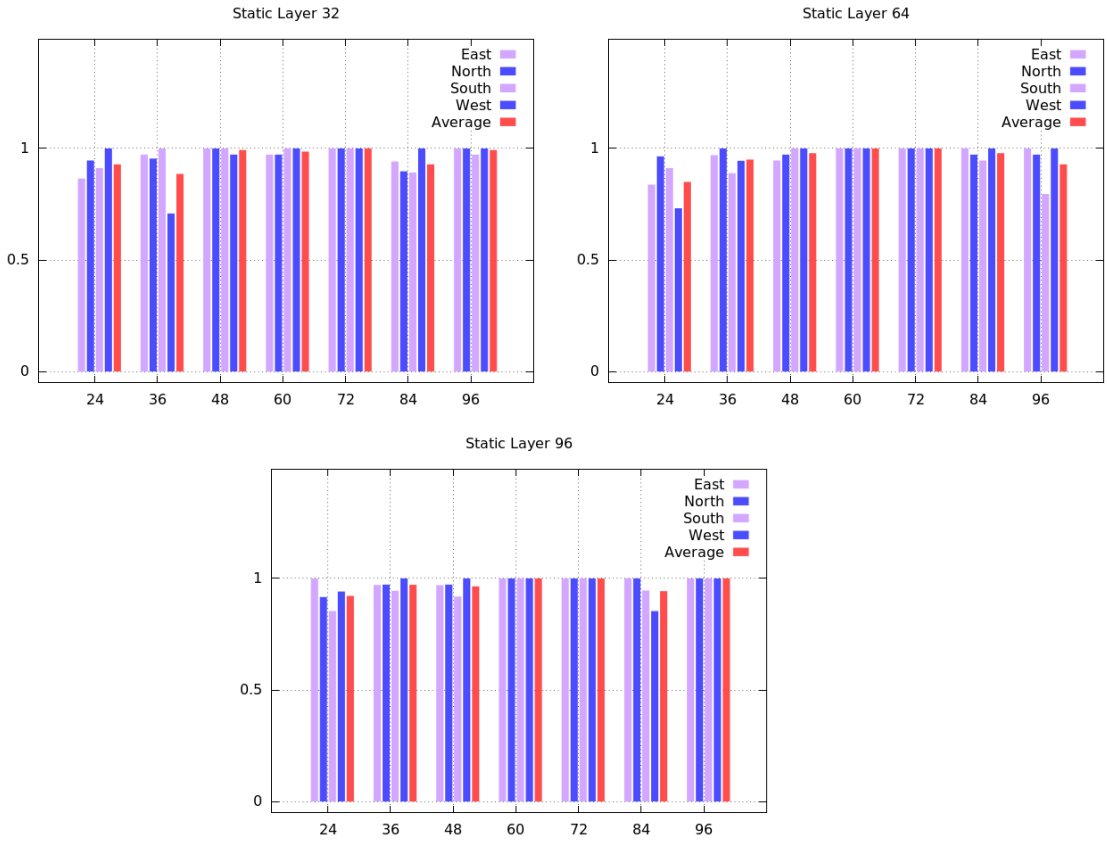


Figure 18: MinMaxScaler Performance in cluttered environment. Most of the topologies exhibit rather good performance.

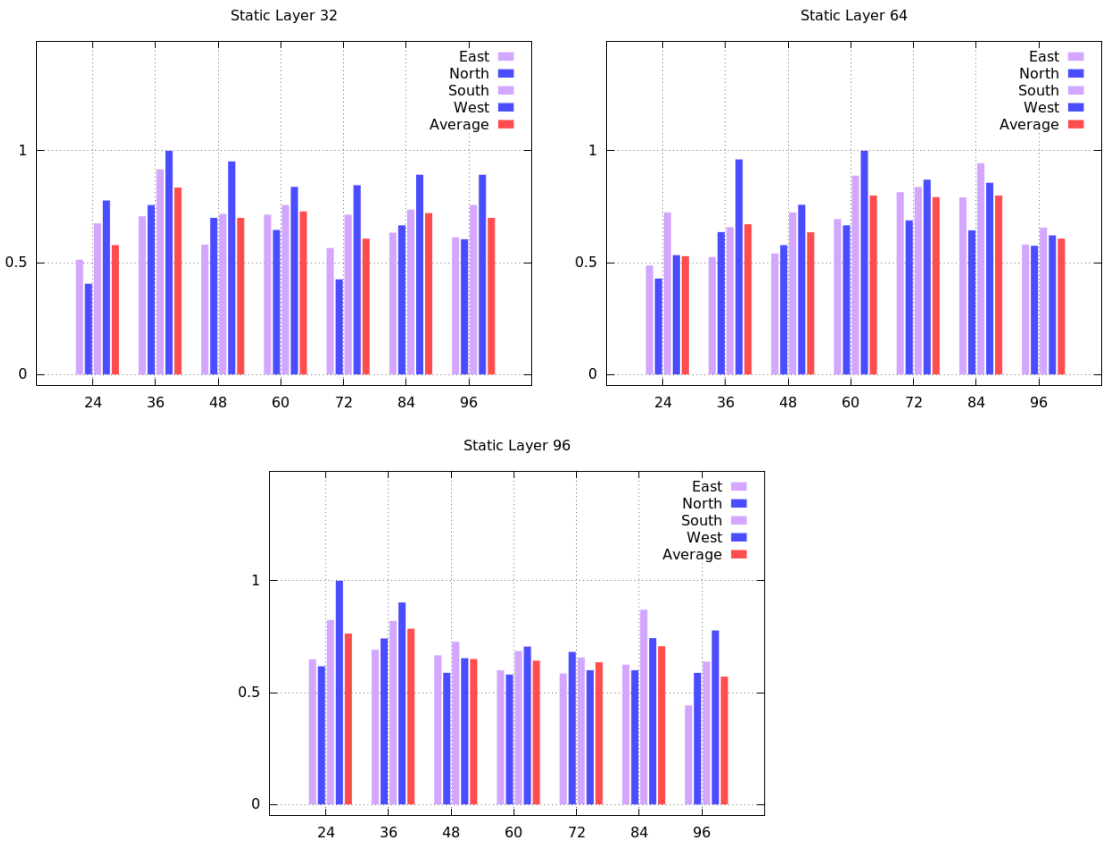


Figure 19: Normalizer Performance in cluttered environment

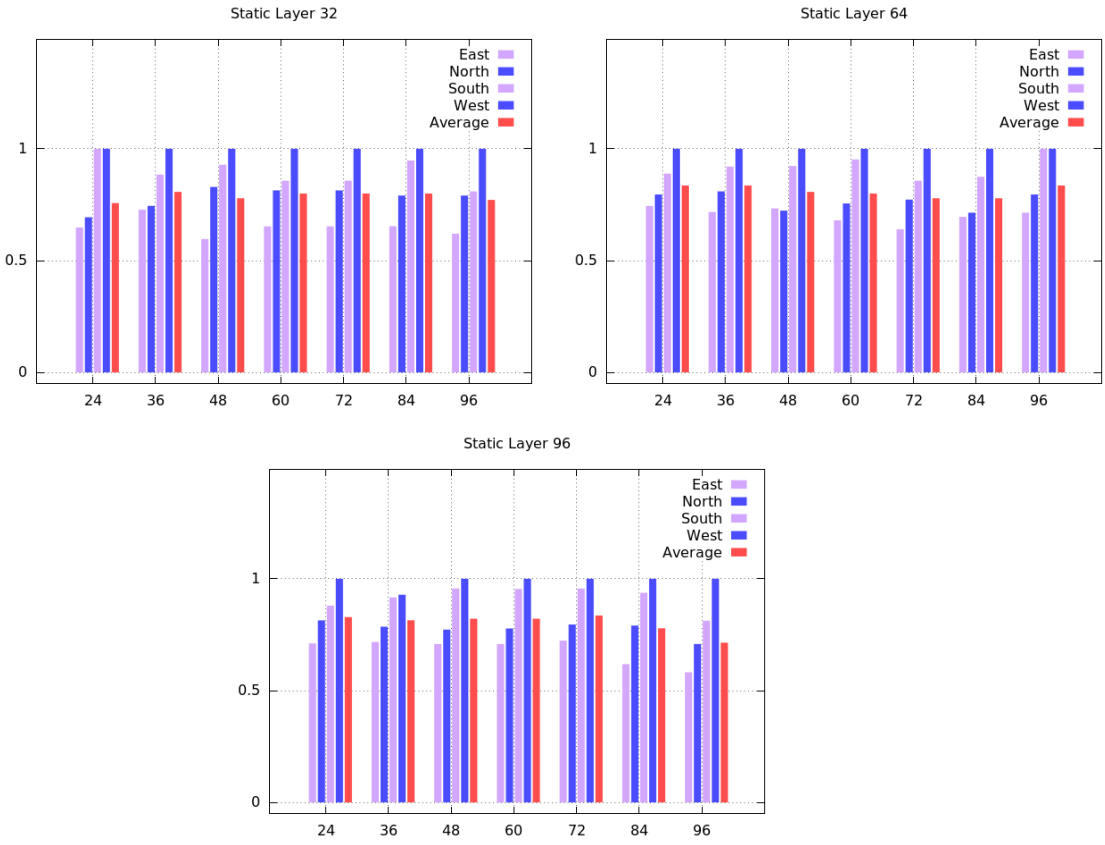


Figure 20: StandardScaler Performance in cluttered environment

5.3.3 Increasingly Cluttered Data Sets

With the previous cluttered data not exhibiting too bad performance with the MinMax scaling, it makes sense to try to introduce even more clutter and see how well the networks can handle the increased complexity. The next tier of data with additional clutter, totaling 10 randomized sources yields even a bigger decline in precision as can be seen on figures 21, 22 and 23.

Yet again the MinMax scaling performs superior to the other variants of scaling, with the majority of the population reaching over 0.9 precision. The normalizer is staying at similar performance as the previous cluttered set. The StandardScaler is yet again at roughly 70% in the majority of cases.

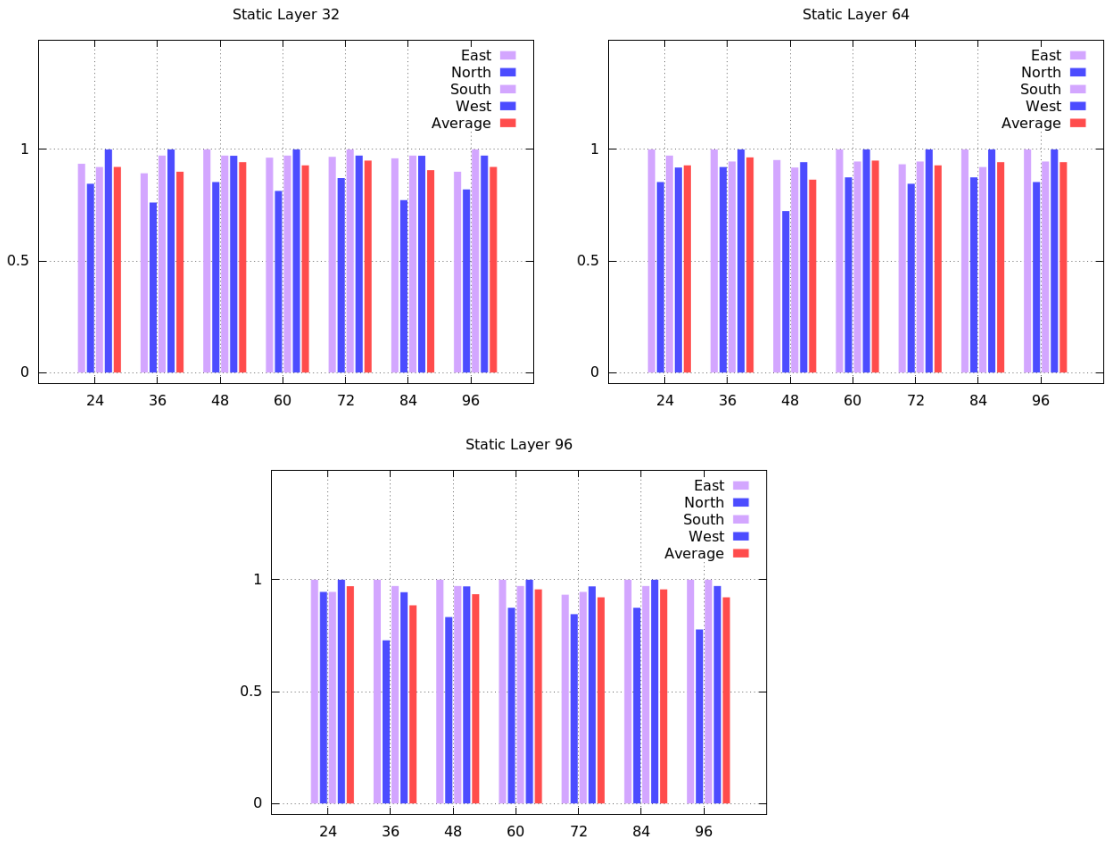


Figure 21: MinMaxScaler Performance in cluttered environment 2

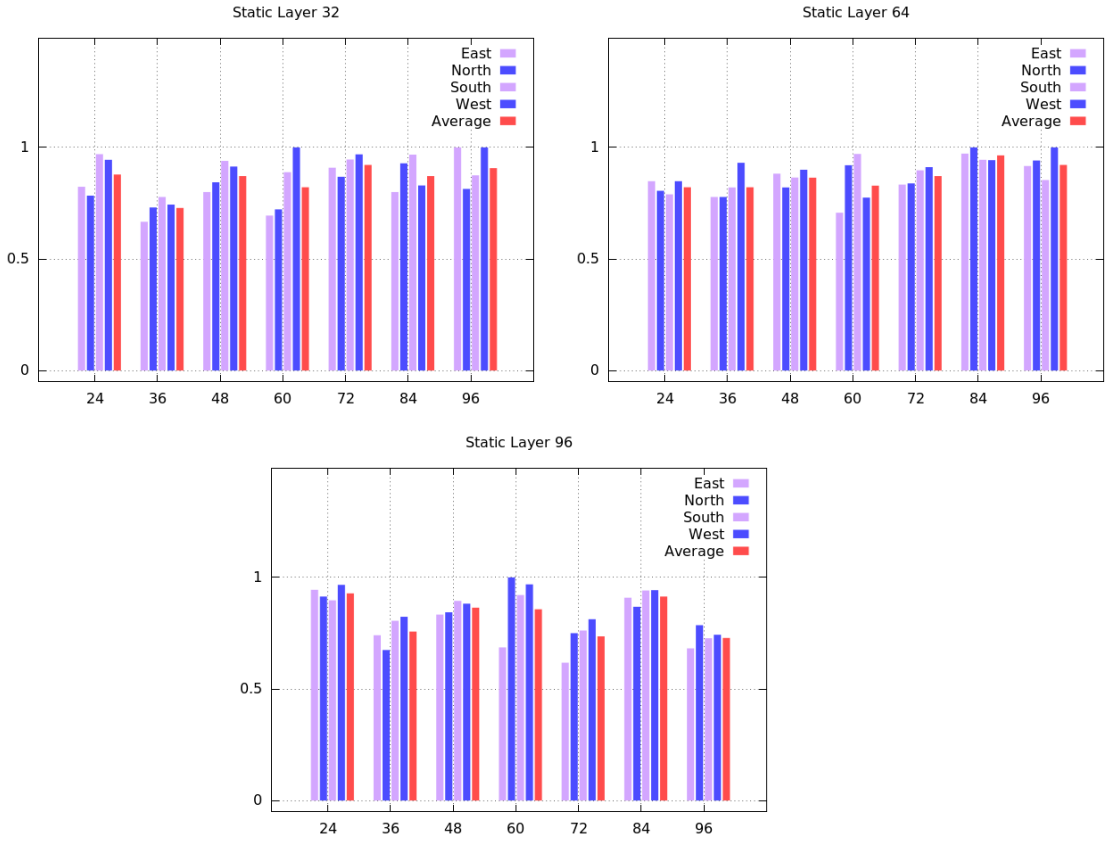


Figure 22: Normalizer Performance in cluttered environment 2

The variations show that while the networks do handle the cluttered samples more gracefully than the simple max approach, their precision is nowhere near what is deemed useful to do a simple detection. Consider the case that with 75% precision 1 out of 4 predictions is wrong, and a wrong prediction in this case is atleast 45 degrees from the target.

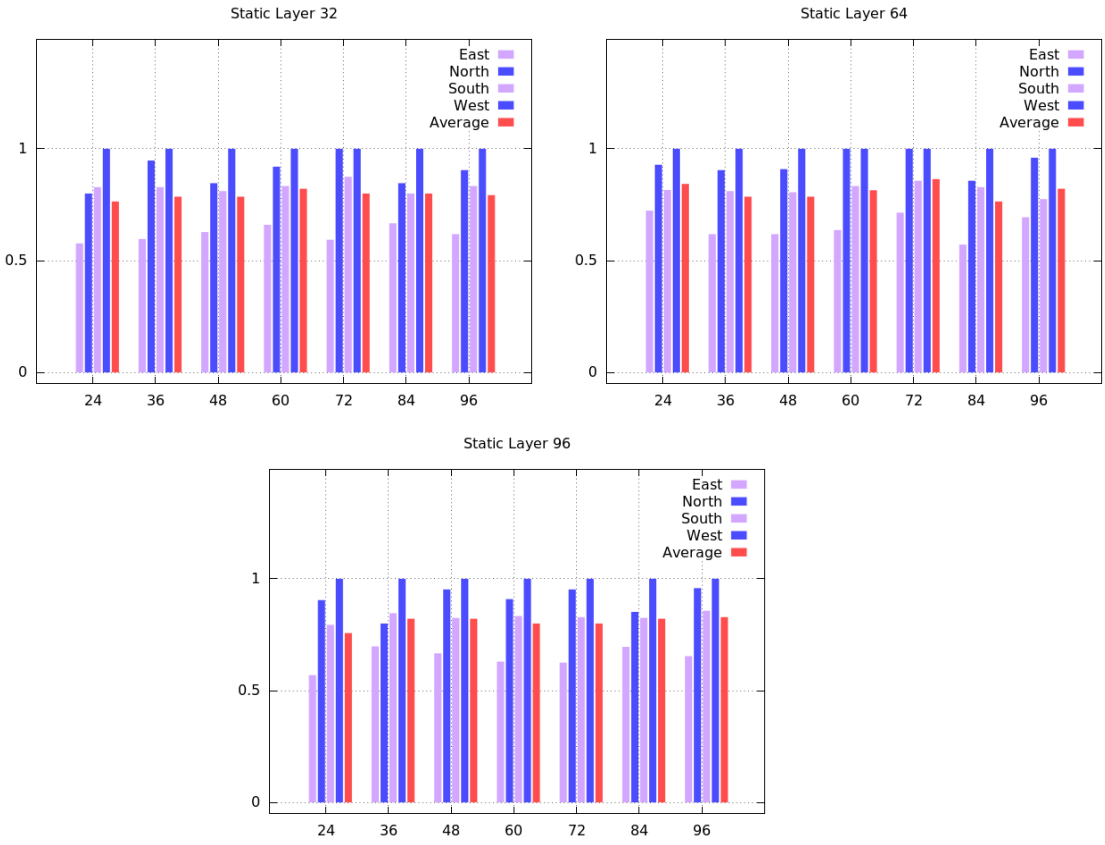


Figure 23: StandardScaler Performance in cluttered environment 2

5.4 Incremental Data Sets

The previously studied cases all derive from one type of training data, which only consist of samples from one variation, ie. only uncluttered or 5 clutter sources.

The next exploration is done using a incremental approach where the initial training is done using uncluttered data combined with the cluttered subsets or given the sets $S_{clutter0}, S_{clutter5}, S_{clutter10}$:

$$S_{i1} = S_{clutter0} \cup S_{clutter5}$$
$$S_{i2} = S_{clutter0} \cup S_{clutter5} \cup S_{clutter10}$$

With the incremental data the results are different from the previous smaller subsets of data, as depicted on figures 24-26.

The tendency is clear though, which is less precision for all 3 pre-processors. The MinMax scaler is starting to drop in performance with the population being around 0.7, while normalizer stays at roughly 0.75 and above. The standard scaler is falling far behind the others and is closer to .5 in precision.

5.4.1 Incremental Set 2

The last set follows the same tendency, however the MinMax scaler seems to produce the highest scores in this case staying at around 0.7 and higher. The normalizer has comparable performance again while the standardscaler ought to be avoided for this particualr problem.

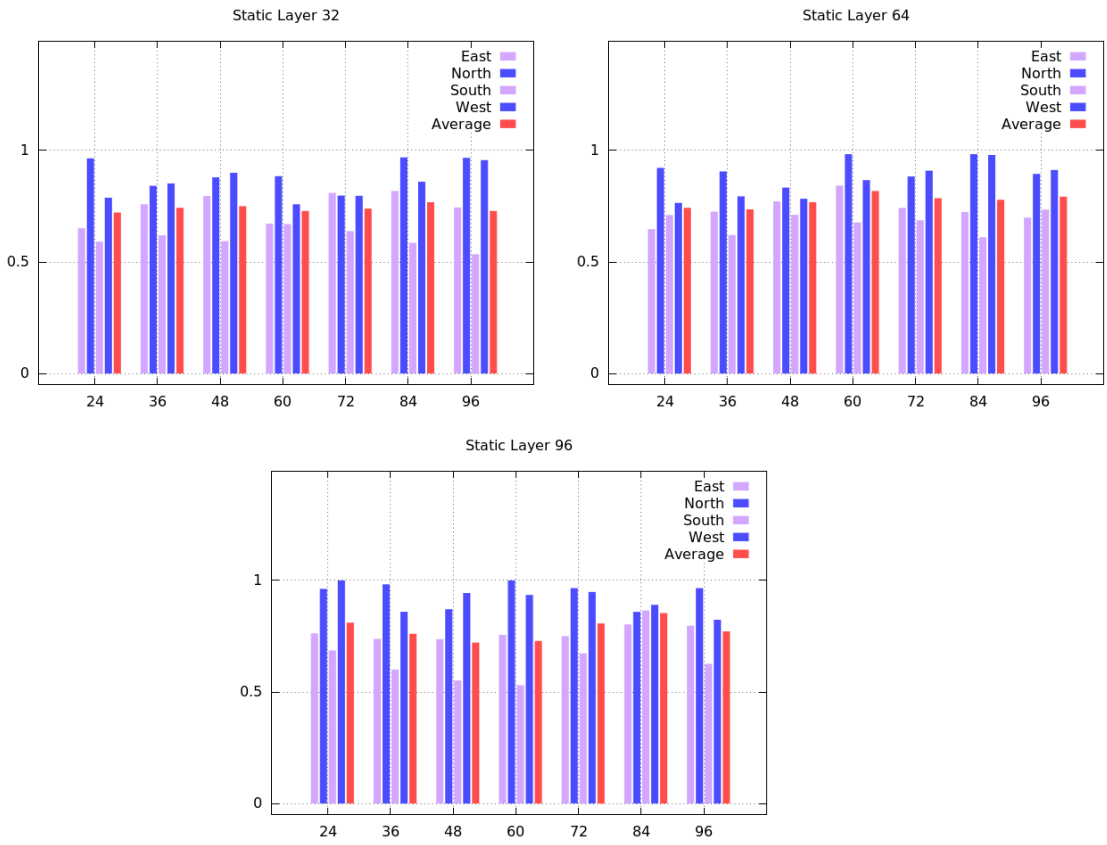


Figure 24: MinMaxScaler Performance using mixed data of noclutter and 5 clutter sources

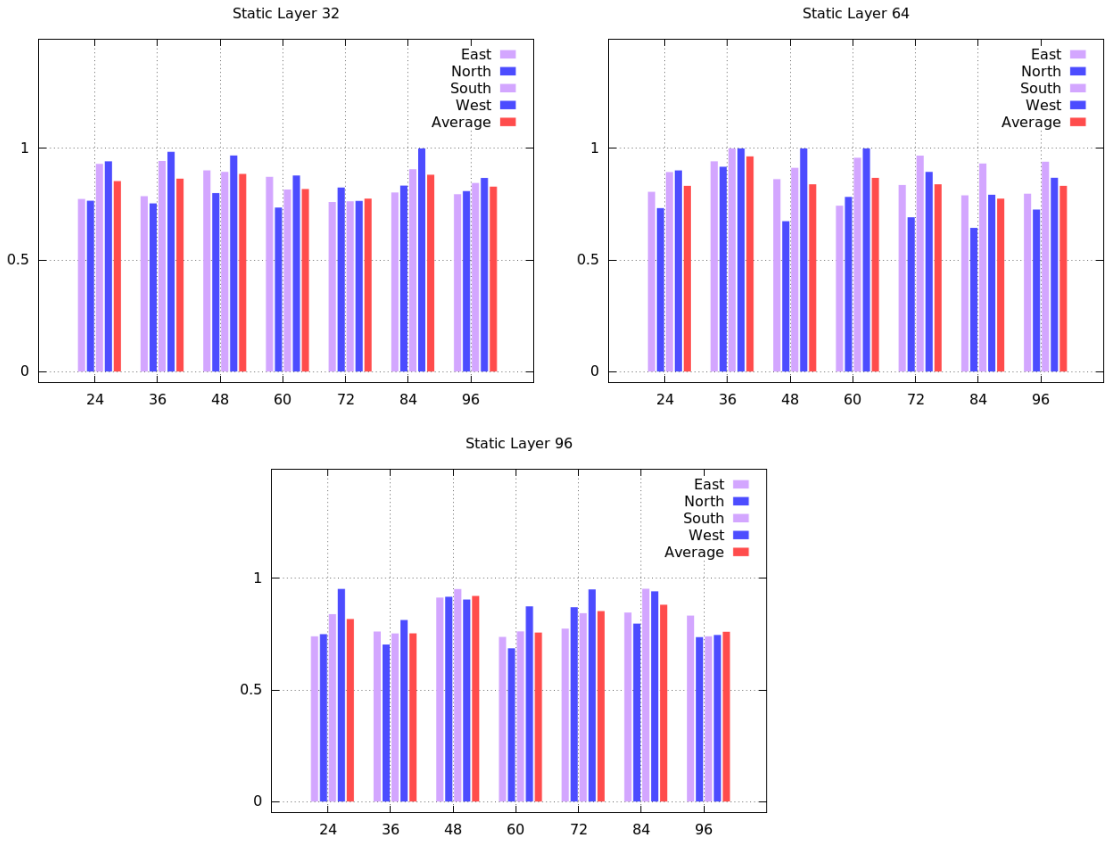


Figure 25: Normalizer Performance using mixed data of noclutter and 5 clutter sources

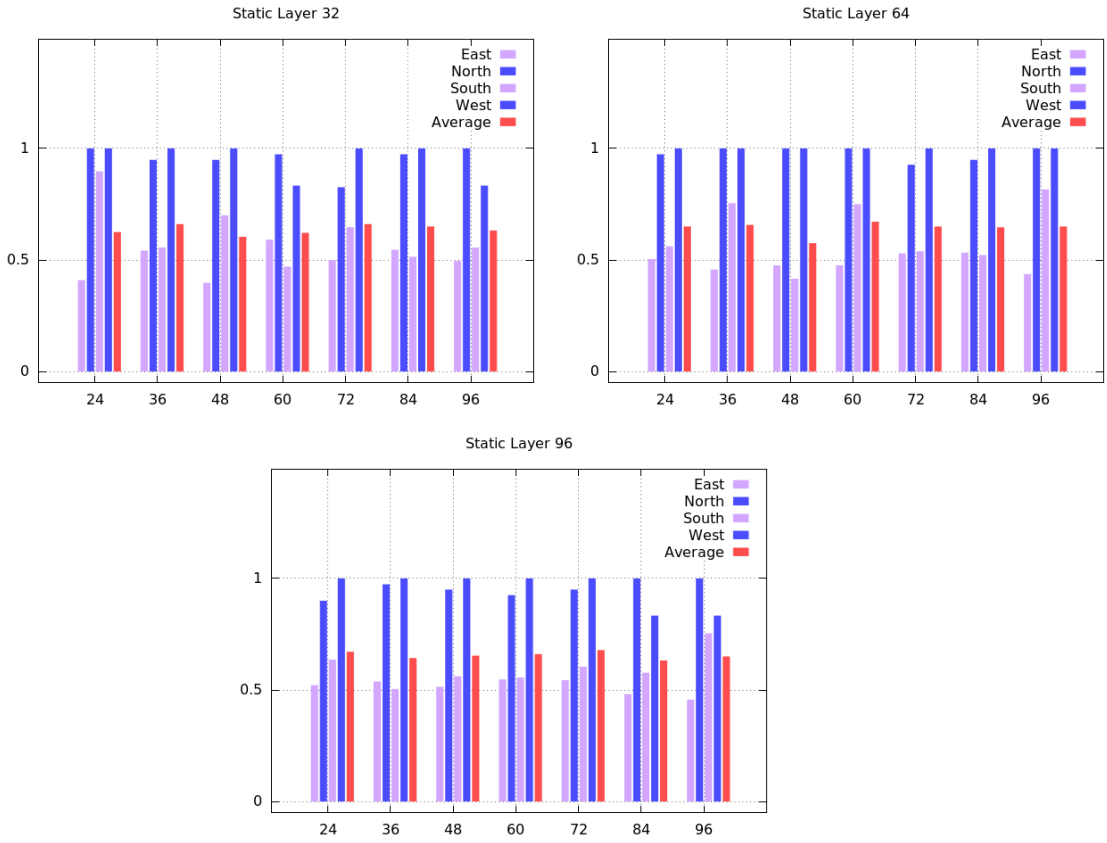


Figure 26: StandardScaler Performance using mixed data of noclutter and 5 clutter sources

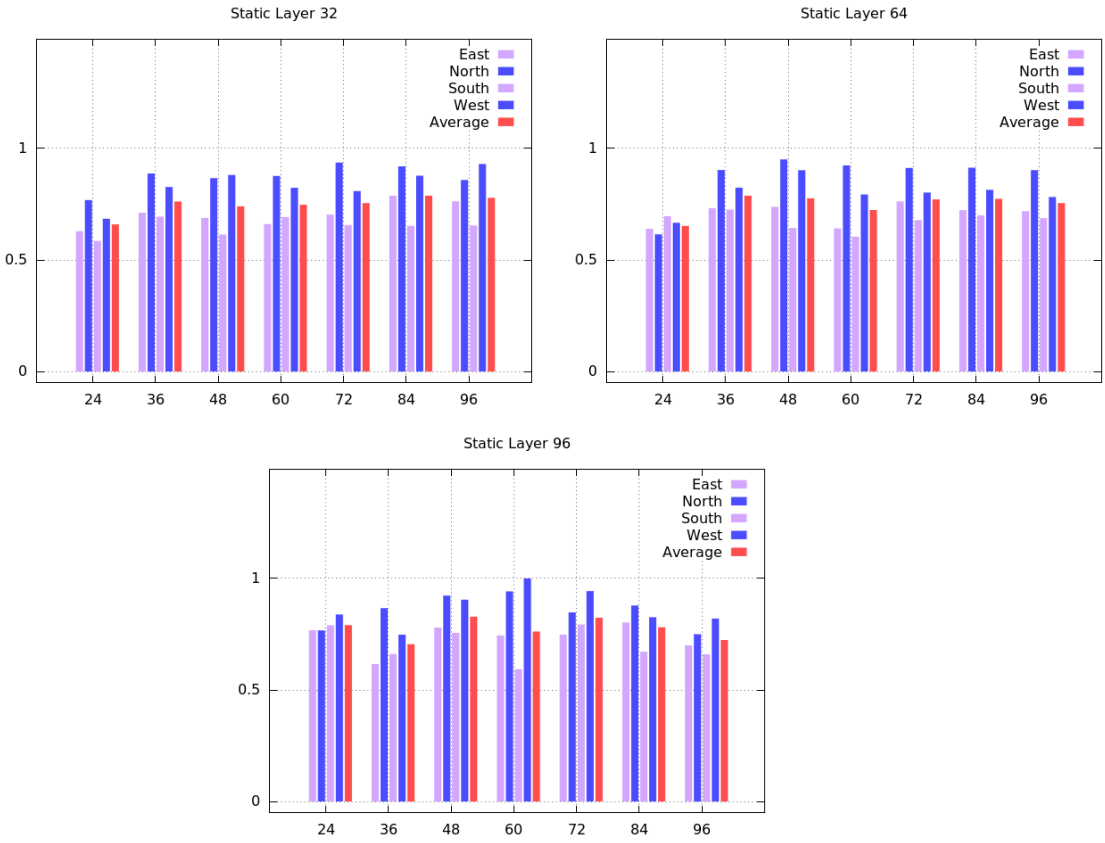


Figure 27: MinMaxScaler Performance using mixed data of noclutter and both 5 and 10 clutter sources

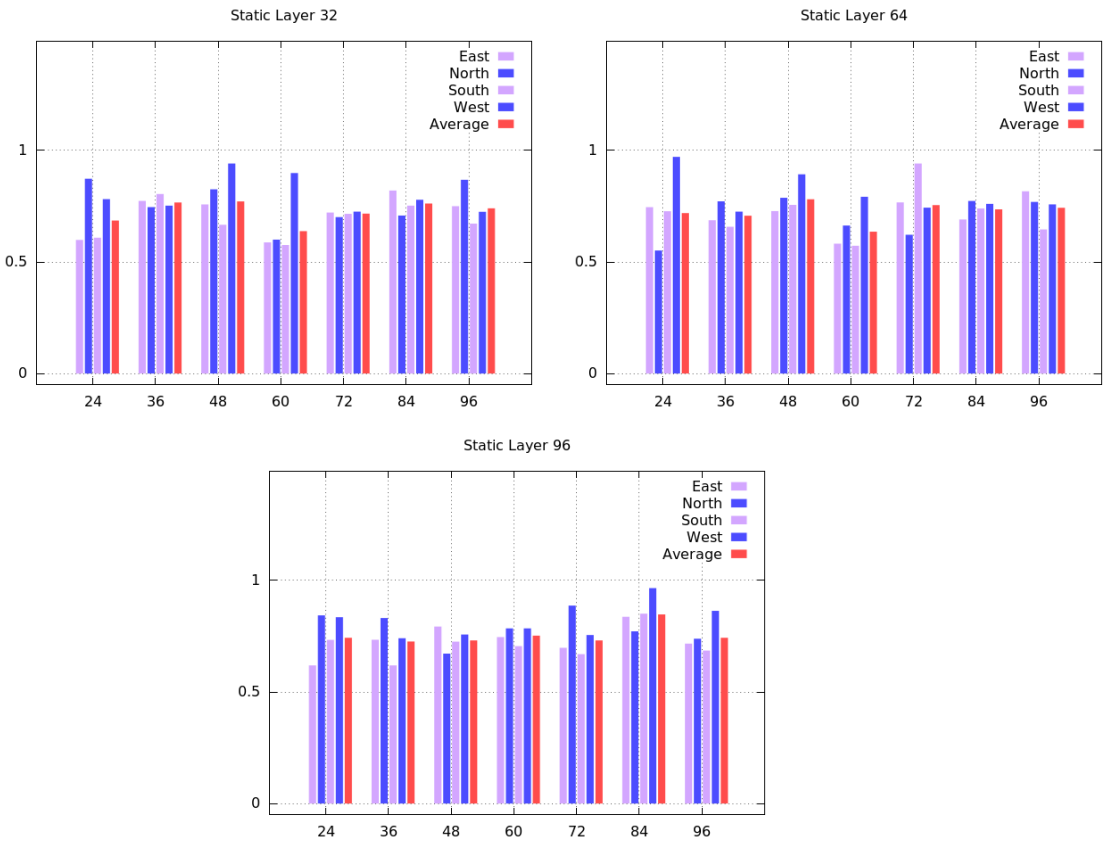


Figure 28: Normalizer Performance using mixed data of noclutter and both 5 and 10 clutter sources

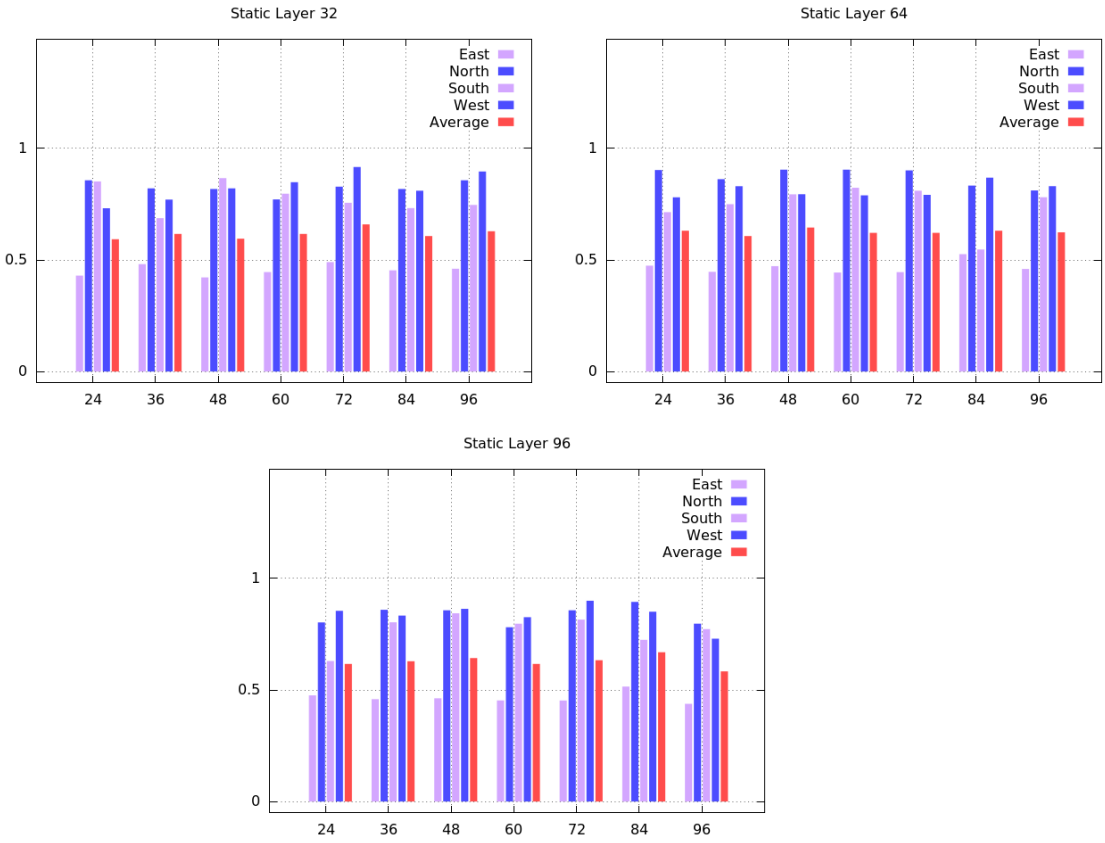


Figure 29: StandardScaler Performance using mixed data of noclutter and both 5 and 10 clutter sources

5.5 Interim Conclusion

While the max_predict performed really poor in most cases with the introduction of cluttered environments and exceptionally well in uncluttered, the inherent nature of RADAR makes the uncluttered environment less likely to occur, hence for the max function to be of any use extensive pre-processing of data is required.

The alternative approach which was tested in this case, the variety of trained MLP network toplogies showed promise. While the general precision of the networks was not impressive, some cases do achieve average cases over 70% for the most cluttered sets. As has already been mentioned in section 4, the preparation of data into angle based values results in much information being discarded, hence a revision of this data handling approach could prove fruitful when attempting to improve precision of the networks.

In addition to tweaking the data preparation, one could also consider replacing binning with filters. This proposed technique of introducing DSP is however worthy of an isolated study.

6 Conclusion

As presented in section 2 there has been a exploration in simulation environments for replicating the physical phenomena of RADAR systems. A somewhat fitting configuration for a medium-range radar system has been mimicked and used for scaling experiments for the ML part. The simulation environment was tested experimentally against the mathematical definitions for signal return as seen in eq. 1 in section 2.1. Despite not matching up it was still used as it exhibited the characteristics that one would expect from RADAR, such as lower power return when increasing range to targets, and symmetric patterns when performing scans with static targets.

The following section 3 explored the limiting factors of noisy data and how to improve quality by utilizing filters. While the exploration might be deemed sparse, it was helpful due to giving an additional angle on how to interpret the raw data from the simulated RADAR system.

Once a standard simulation environment was defined and different simulations executed it became apparent that data preparation was of great importance. An exploration in how to sample and prepare data for use with ML was conducted, during which was realized that once the amount of data increases, it is of great importance to make smart decisions in software design such that the processing time is kept efficient.

This also emphasized the point that one should be smart about the libraries and tools one selects for performing tasks, as poorly made decisions and bad utilization can result in poor performance as seen in the data processing python script 'permute.py' and its predecessor 'naive_permute.py'.

The final stage explored how efficient MLPs can be when network topology is varied. Additionally it explored how much cluttered data effects the precision of a network. It was also discovered that the cluttered data was generated with a bias, and this bias affected every test made. This means that both the max prediction and the networks need to be tested further to be able to determine something more general.

The different MLPs performances have been compared to a simple way of processing the responses. The comparison led to the realization that in cluttered data the MLPs can achieve better performances than the baseline max function if tuned sufficiently. In uncluttered data the max predict is just as

good as the majority of the networks, and in some combinations the networks can achieve worse performance than the baseline.

Shortcomings

While the study in ML did show some promise when evolving the topologies it needs refinement. First and foremost one should design more complex experiments such that the networks can be trained to something that transcends their current state of, at best, an silly application on a complex domain.

While the amounts of data have been upscaled and many samples have been produced, the limitation of only producing data for 4 directions, rather than the full 360 degrees are insufficient. Producing samples for the cases in between the current 4 directions would be a welcomed addition to the current data sets. In general producing more diverse data is a welcomed addition to cover additional cases.

7 Future Work

This project can be broken down into some major blocks where each can be improved in some way.

Simulation While the FERS simulator was able to simulate some radar scenarios, it is not the most accurate simulation environment. Hence it makes sense to investigate alternatives to the FERS simulator, in particular an application of the matlab Phased Array Systems Toolbox, as this can be a stepping stone to more complex radar configurations with ease. Alternatively i discovered GNU Radio, which is able to simulate radio signals. A plugin is being built for the GNU Radio toolkit which is aimed specifically for the exploration of the RADAR domain. The previously mentioned plugin, GNU Radio Radar Toolbox[3], might be interesting to explore seeing its open source.

Digital Signal Processing The DSP field is broader than first anticipated. A more thorough exploration of filtering and other signal processing approaches can be very interesting to potentially push the boundaries of how well neural networks can perform for this particular application.

Machine Learning Investigate the use of MLPs to recognize multiple targets. The current investigation was rather one sided, due to complications during the early stages of the project. And the first vision was to actually be able to establish tracks and distinguish multiple targets of interest from one another, however in reality this was harder than first anticipated. Nevertheless it still makes a interesting topic to explore in the future.

References

[1] Marc Brooker. "The Design and Implementation of a Simulator for Multistatic Radar Systems". 2008. URL: http://www.rrsg.uct.ac.za/theses/phd_theses/mbrooker_thesis.pdf.

[2] *Flexible Extensible Radar Simulator*. URL: https://sourceforge.net/projects/fers/.

[3] *GNURadio Radar Toolbox*. URL: https://github.com/kit-cel/gr-radar.

[4] Ian J. Goodfellow et al. "Multi-digit Number Recognition from Street View Imagery using Deep Convolutional Neural Networks". In: *CoRR* abs/1312.6082 (2013).

[5] Jiawei Han. *Data Mining: Concepts and Techniques*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2005. ISBN: 1558609016.

[6] *MATLAB Programming/Differences between Octave and MATLAB*. URL: https://en.wikibooks.org/wiki/MATLAB_Programming/Differences_between_Octave_and_MATLAB.

[7] *Pandas Library*. URL: http://pandas.pydata.org/pandas-docs/stable/.

[8] G. K. Prasad and J. S. Sahambi. "Classification of ECG arrhythmias using multi-resolution analysis and neural networks". In: *TENCON 2003. Conference on Convergent Technologies for Asia-Pacific Region*. Vol. 1. 2003, 227–231 Vol.1. DOI: 10.1109/TENCON.2003.1273320.

[9] *Scikit Learn*. SKLearn. URL: http://scikit-learn.org/stable/documentation.html.

[10] *Scikit Learn - Supervised Neural Networks*. SKLearn MLP. URL: http://scikit-learn.org/stable/modules/neural_networks_supervised.html.

[11] *SciPy*. Python Science lib, Signal Processing. URL: https://docs.scipy.org/doc/scipy/reference/signal.html.

[12] Merrill Ivan Skolnik. *Introduction to RADAR systems*. McGraw-Hill, 2001.

[13] *Stream Editor, sed*. URL: https://www.gnu.org/software/sed/manual/sed.html.

[14] *Web resource for RADAR systems*. URL: http://www.radartutorial.eu/html/author.en.html (visited on 2016).

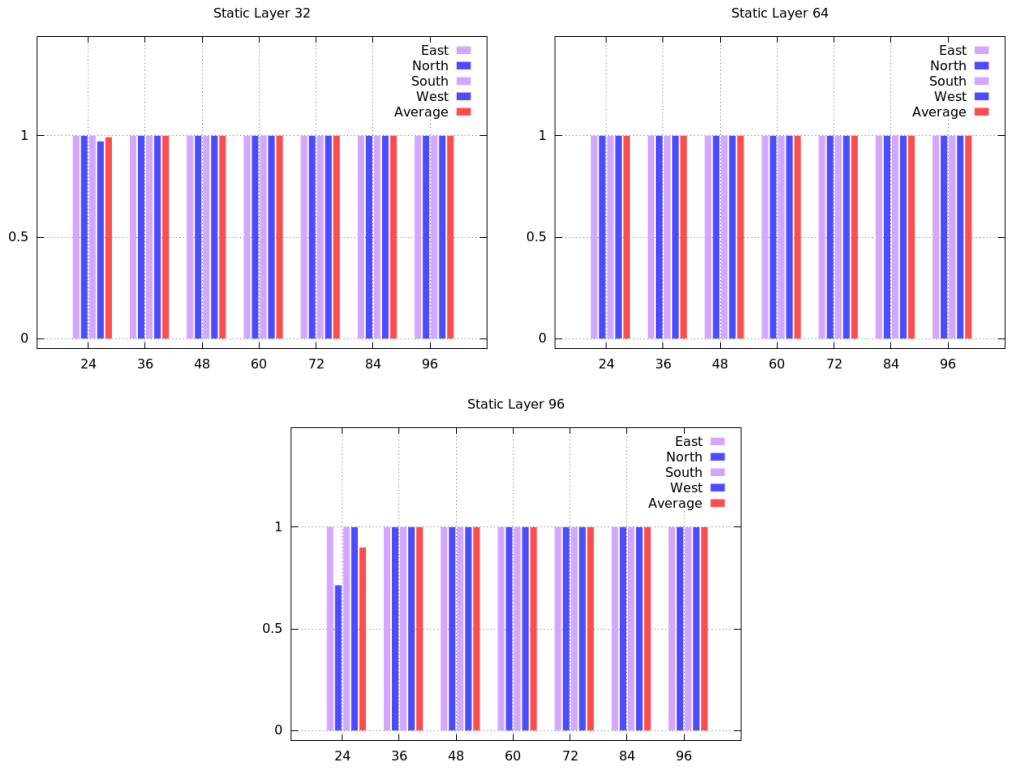
[15] *Windows Subsystem for Linux documentation*. URL: https://msdn.microsoft.com/en-us/commandline/wsl/about.

[16] *Xmllint overview*. Alternatively use man xmllint in the shell. URL: http://manpages.sgvulcan.com/xmllint.1.php.

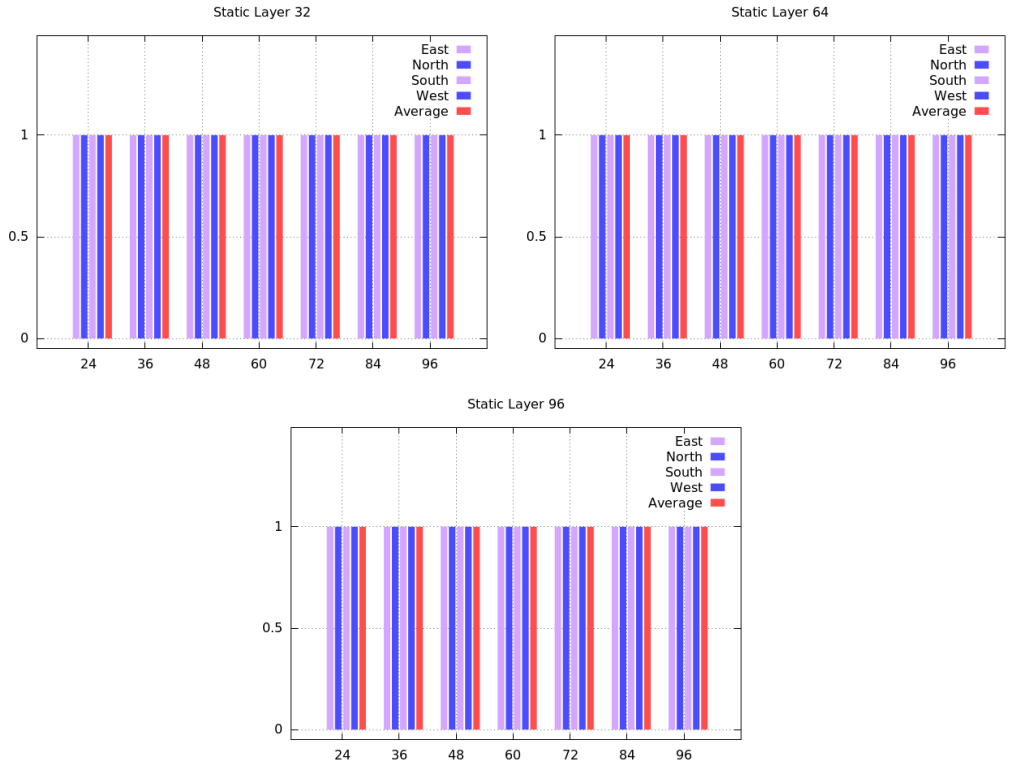
Appendix A Uncluttered Data

Examples of precision with uncluttered data using different scalers. The static layers are of sizes 36, 64 and 96 where the variable layers are ranging from 24 to 96.

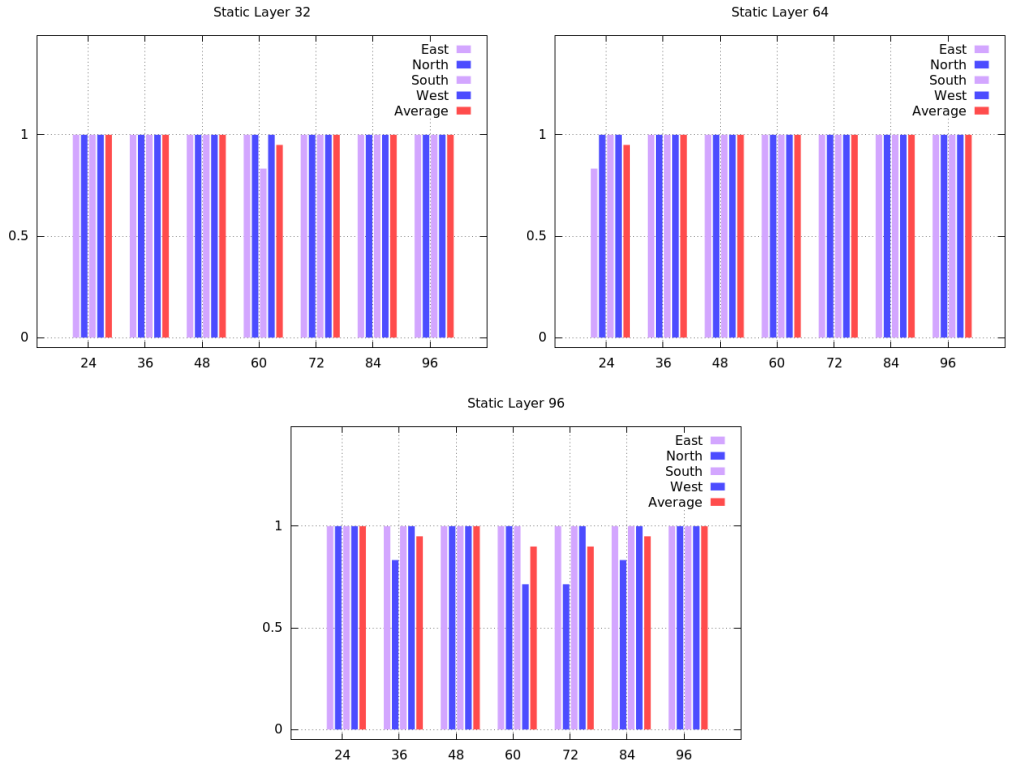
A.1 Min Max Scaling



A.2 Normalizer



A.3 Standard Scaling



Appendix B Repository

The code and some of the sample data has been made readily available online at a public Github repository.

https://github.itu.dk/inau/ml_to_radar

Appendix C Max Prediction Code

C.1 predict

Listing 8: code/max_predict.py

```
8  # CSV file containing sample
9  # length per scan in seconds
10 # returns [sequence of data], predicted_direction
11 def predict(file, scan_t):
12     data = pd.read_csv(file, header=None)
13     data.columns=('t','p','ph','d')
14 #    print('%s loaded' % file)
15     t0= math.floor(data.iloc[0]['t']) # round to nearest int
16     tMx = t0 + scan_t
17     idx = data['p'].argmax()
18 #    print('%i id contains max bounds %f to %f' % (idx,t0,tMx))
19     t = data.iloc[idx]['t']
20 #    print('%f is time stamp' % t)
21     dir_q = (scan_t/4.0) #1/4 of a rotation
22     dir_o = (scan_t/8.0) #1/8 of a rotation
23     lb=t0+dir_o
24     ub=tMx-dir_o
25     def ret(dir):
26         return (data, dir)
27 
28     if t <= lb or t >= ub:
29         return ret('east')
30     else:
31         if t <= lb+dir_q:
32             return ret('north')
33         else:
34             if t <= lb+(2*dir_q):
35                 return ret('west')
36             else:
37                 return ret('south')
```

C.2 predict statistics

Listing 9: code/max_predict.py

```
48 def predict_stats(root, out = ''):
49     # open root folder
50     sourcedir = os.path.dirname(root)
51     # if directory exists open sources, else open single file
52     if os.path.exists(sourcedir):
53         print('Source␣dir␣exists..')
54         source_folders = os.listdir(sourcedir)
55         print('Sources␣found..␣%s' % (source_folders))
56 
```

```
57          results = []
58          for folder_name in source_folders:
59              positive = 0.0
60              total = 0.0
61              print('- traversing %s' % folder_name)
62              dirs = glob.glob('%s%s/*.csv' % (root,folder_name))
63              for csv in dirs:
64                  #                print('- - file %s' % csv)
65                  total += 1
66                  d,l = predict('%s'%csv, 2)
67                  #                print('Sample from %s predicted as %s' % (
                        folder_name, l))
68                  if folder_name == l:
69                      positive += 1.0
70              results.append( (folder_name, positive, total) )
71
72          old = sys.stdout
73          if(out != ''):
74              sys.stdout = open(out, 'w')
75          print('#Label        correct      total        percentage')
76          for r in results:
77              l, p, t = r
78              print('%s        %s       %s       %f' % (l,p,t,(p/t)))
79          sys.stdout = old
```