

University of Alabama in Huntsville

LOUIS

Theses

UAH Electronic Theses and Dissertations

2024

ClearScan : a machine learning system for customized, site-specific radar image filters

Erick Jones

Follow this and additional works at: <https://louis.uah.edu/uah-theses>

Recommended Citation

Jones, Erick, "ClearScan : a machine learning system for customized, site-specific radar image filters" (2024). *Theses*. 683.

<https://louis.uah.edu/uah-theses/683>

This Thesis is brought to you for free and open access by the UAH Electronic Theses and Dissertations at LOUIS. It has been accepted for inclusion in Theses by an authorized administrator of LOUIS.

**CLEARSCAN: A MACHINE LEARNING
SYSTEM FOR CUSTOMIZED,
SITE-SPECIFIC RADAR IMAGE FILTERS**

Erick Jones

A THESIS

**Submitted in partial fulfillment of the requirements
for the degree of Master of Science**

in

The Department of Computer Science

to

The Graduate School

of

The University of Alabama in Huntsville

August 2024

Approved by:

Dr. Huaming Zhang, Research Advisor/Committee Chair

Dr. Jacob Hauenstein, Committee Member

Dr. Deepak Acharya, Committee Member

Dr. Letha Etkorn, Department Chair

Dr. Rainer Steinwandt, College Dean

Dr. Jon Hakkila, Graduate Dean

Abstract

CLEARSCAN: A MACHINE LEARNING SYSTEM FOR CUSTOMIZED, SITE-SPECIFIC RADAR IMAGE FILTERS

Erick Jones

**A thesis submitted in partial fulfillment of the requirements
for the degree of Master of Science**

Computer Science

The University of Alabama in Huntsville

August 2024

In the field of radar meteorology, a perpetual problem is removal of so-called anomalous propagation (AP), *i.e.*, non-precipitation echoes, from the produced images. Much work has been done in this area already, including conventional heuristic algorithms as well as machine-learning systems such as neural networks. Often the focus is on certain familiar radar architectures such as WSR-88D, also known as NEXRAD. However, a large number of radars exist which are not identical to NEXRAD; and there are also environmental differences such as RF interference which can affect the success rate of existing AP removal strategies. The focus of this paper is to present a flexible machine-learning system for this task which provides a convenient training interface so it can be adapted to the specific conditions present at any given radar site to create a customized filter. We have named this system ClearScan.

Acknowledgements

I would like to acknowledge Baron Weather for providing facilities, equipment, funding, and other support including encouraging other employees to contribute and participate.

I would also particularly like to thank Sherman, my department lead, who encouraged me to pursue this project and has often been at least as enthusiastic about it as I have. And my co-worker Emily, who has done the most hands-on work of using this software and providing valuable feedback.

Finally, I would like to thank my advisor, Dr. Huaming Zhang, for cluing me in on how to do a thesis.

Table of Contents

Abstract	ii
Acknowledgements	iv
Table of Contents	vi
List of Figures	vii
Chapter 1. Introduction	1
1.1 Background Overview	2
1.2 Motivation	3
1.3 Methodology	3
1.4 Thesis Organization	6
Chapter 2. Background and Related Work	7
Chapter 3. ClearScan System Operation	13
3.1 Sample Import	13
3.2 Labeling	15
3.3 Network Builder	17

3.4 Test Builder	20
Chapter 4. Implementation Details	23
Chapter 5. Case Study: Bangladesh Radar	28
5.1 Introduction	28
5.2 Dual Networks	29
5.3 Results	30
5.4 Alternative Network Comparison	30
5.5 Quantitative Results	32
5.6 Potential Drawback	34
Chapter 6. Conclusion	39
6.1 Contribution	39
6.2 Future Work	39
References	41

List of Figures

3.1	Datasets with navbar	13
3.2	Dataset import menu	14
3.3	Dataset page	15
3.4	Labeling interface	16
3.5	Network editor	17
3.6	Network builder overview	18
3.7	Network training	18
3.8	Network training completed	19
3.9	New network added	20
3.10	Test builder overview	20
3.11	Editing the list of test samples	21
3.12	Selecting a test to run	22
3.13	Results column for Full Test shown.	22
5.1	Bangladesh radar with contamination	28
5.2	RF labeled image	30
5.3	Plain AP labeled image	31
5.4	Bangladesh radar after cleanup	32
5.5	Alternative filter	33
5.6	Bangladesh filter test scores	33
5.7	Bad precip case	35
5.8	Bad filter result	36
5.9	Bad PHI (Differential Phase) data	37

5.10 Good PHI (Differential Phase) data	38
---	----

Chapter 1. Introduction

For purposes of real-time weather detection, modern meteorologists rely more on radar than on any other sensing equipment. This is increasingly true in recent years as radar coverage has increased, and as the technology has improved – particularly with the introduction of dual-polarization radars [7]. Yet, the nature of this equipment’s operation is such that some of the echoes detected do not correspond to actual weather [10]. Temperature inversions, sun spikes, radio-signal interference, and even wind farms can add clutter to the returned image. A frequently studied problem, then, is how to automatically distinguish between these anomalous echoes and those corresponding to actual weather, so as to present a clean image showing only precipitation – either to be displayed directly, or to be used in further processing steps such as rainfall accumulations.

When working with computers, the most familiar mode of operation is where we tell the computer what to do and how to do it, and it performs as instructed. Imagine if, instead, the computer could figure out *for itself* how to do what we ask. The field of machine learning seems to hold a glimmer of this possibility – this idea that the computer becomes a “partner” in solving the problem. We may not yet be at the point where we can just speak conversationally with

the computer like in Star Trek; but it is always worth a try to see what problems we can solve with machine learning, if only to further develop the concept.

1.1 Background Overview

The idea of using machine learning and neural networks with the particular problem of cleaning up radar images is certainly nothing new. A number of papers have already been published on the subject, for example:

- [10] presents a very simple fully-connected, feed-forward neural network to make a decision on a per-pixel basis – but with a robust set of pre-processed inputs, including several that were computed using a “neighborhood” of each pixel.
- [14] instead takes the approach of stacking multiple radar images taken at different vertical “tilts” to produce a three-dimensional radar image; and then running a 3D convolutional neural network against this image volume.

(Further examples of related work in this area can be found in Chapter 2.)

There have also been conventional (non-machine-learning) algorithms developed; and dual-polarization technology in particular has enabled some successful hydrometeor classification schemes [4].

However, as will be discussed in the next section, these existing tools do not necessarily work with radars that have different operating conditions than those for which they were developed.

1.2 Motivation

A challenge that we have often observed at radar customer sites is that the radar is different enough from NEXRAD that off-the-shelf solutions do not work as well. In particular, NEXRAD radars work with S-band frequencies, but many commercial entities that own radars must use C-band, since S-band is reserved for government use. Additionally, when radars are deployed in other countries, they sometimes find that RF interference is substantial since frequencies are not regulated as tightly as they are in the U.S.

Our objective, then, was a system that would help radar users create custom AP filters – enabling them to train it with their own data, under their own conditions. In the spirit of *computers acting in partnership with users*, we decided that machine-learning technology provides a sound basis for the type of filter training environment that we envisioned.

1.3 Methodology

In order to provide the most flexibility, the basic structure used is a convolutional neural network (CNN) with two hidden layers. Radar data processing is, at its core, an exercise in image processing; so it stands to reason that techniques such as CNN which are often used in image processing would be applicable here.

Rather than combine with a fully-connected network (FCN) to produce a single answer for the whole image, we opted to use just the CNN part; this way, decisions for multiple pixels in the image could be generated in a single run. Only

the local neighborhood around a pixel (determined by the size of the filters) will contribute to the decision reached for that pixel.

Since dual-polarization radars are becoming more common, we decided to require the dual-pol data moments (differential reflectivity, correlation coefficient, and differential phase) as inputs in addition to the echo intensity (“reflectivity”). This does somewhat limit the applicability of the method since it cannot be used as-is with single-polarization radars. Extending these techniques is beyond the scope of this paper (but may be a future project).

Unlike [10], we opted not to have any special pre-processing. Undoubtedly, the robust set of pre-processed input variables used there represents a great deal of expertise from the field. For our purposes, we believe it makes more sense for the network to decide for itself what patterns are applicable; so only the raw data moments from the radar are provided as input.

The training interface provided is web-based, and designed for ease of use; but also includes all the elements that are needed to guarantee a robust machine-learning implementation. In particular, the training interface (which will be described in more detail in the next section) provides:

- Selection of sample data by date/time, with automatic retrieval from a historical data archive for purposes of labeling and testing.
- A visual, paint-program-like interface for labeling the data.
- A network builder that lets you select which samples to use for training and for validation, as well as tune meta-parameters such as number of filters in the hidden layers.

- Online network training, which shows the test and validation success rates as it goes along, and provides a “stop” button that can be used once the operator decides that sufficient training has taken place.
- A test definition interface, where you select a collection of labeled samples to use as a test and then run previously trained networks to see how they perform.
- Multi-user support, and facilities for entering comments that are visible to other users.

1.4 Thesis Organization

In the next chapter, we will provide some background with examples of related work in the field of machine-learning techniques for radar image quality improvement. Chapter 3 will introduce the filter training system we have created and offer a description of the user interface and operation. In Chapter 4, we provide some of the more interesting implementation details to help clarify how it works. Chapter 5 describes, as a case-study, our experience with deploying this system for a radar in Bangladesh. Chapter 6 concludes with a summary and some ideas for future work.

Chapter 2. Background and Related Work

Research and experimentation on the problem of detection and removal of non-precipitation echoes from radar images goes back decades. For example, an article by K. A. Browning from 1978 expands on the idea of examining the “fluctuation rate of the radar return” as a discriminating feature [3], in turn crediting an earlier report from 1975 for the basic idea [9].

The idea of applying machine learning to the task likewise has a long history. Although the field of ML has itself developed rapidly in recent years to make more advanced techniques available, one can nonetheless find work in previous decades that tried to make the best use of the tools available at the time. In some cases that meant looking for less granular results (such as classifying whole images or regions instead of individual pixels), or making heavy use of pre-processing techniques to achieve a reduction in the dimensionality of the inputs.

In this chapter, we will provide summaries of the most interesting papers we have come across. In comparing the approaches, we find that, aside from the different AI methods used, it is also interesting to see the variation in terms of:

- use of Doppler velocity data;
- use of dual-polarization data;
- use of volumetric a.k.a. 3D data, requiring scans at different elevations;

- pre-processing/feature extraction versus “pure” machine learning.

We will list these in chronological order to give a sense of the progression in this area of research.

- In an article by Grecu and Krajewski from 1999, we see an application of artificial neural networks to the problem. This was an attempt to make use of a large body of archived images from older radars without Doppler capabilities for climatological research purposes.

The author notes that “pixel-by-pixel classification based on reflectivity information from a single scan does not seem possible”, given the applicable state of radar technology as well as that of machine learning; so the paper instead focuses on categorizing *whole images* in terms of the suspected amount of image contamination from AP (*i.e.*, so you can just throw out the most contaminated images). Preprocessing heuristics were used to summarize the input images, so as to feed only 12 inputs into the neural network itself [6].

- Grecu and Krajewski also published a second article that same year, this time tackling the problem of per-pixel classification. They still chose not to use Doppler information, though they mention other contemporary researchers who were starting to incorporate Doppler velocity data.

The technique described makes heavy use of feature extraction (this time computed on small windows around the pixel to be classified), and makes significant assumptions about the data available. In particular, it requires

consecutive scans at the same level in order to estimate "movement" via image correlation, and scans at different elevation to determine the height of the echoing region. Once again, the neural network is small and is highly dependent on effective local feature selection.

They also made the interesting choice that training data would be provided in terms of whole images which are either all precip or all non-precip – even mentioning that this could result in the network having a hard time learning to recognize precipitation echoes embedded inside areas of AP [5].

- The Lakshmanan paper (mentioned in the introduction) is from 2007, and is a lot like the per-pixel classifier from Grecu and Krajewski (and indeed cites the latter). Like Grecu, a list of subjectively determined feature extraction heuristics is used; and volumetric data (scans at multiple elevations) is required.

Doppler velocity is used here, but conditionally. It is noted that the effective range for accurate velocity determination is shorter than that for reflectivity; so they decided to train two networks, one with Doppler velocity data and one without.

One innovation offered here is that the actual usefulness of each of the possible input features is formally determined – basically by trying the network both with and without the given input feature to see if it has any effect on the overall performance [10].

- In 2008, Rico-Ramirez *et al.* described classifiers based on Fuzzy Logic and Bayes methods, comparing the results obtained with each.

Here we see the use of dual-polarization data moments (which would've been relatively new technology and not available everywhere – see [7]).

Also, instead of having a long list of complex features, this offers a more straightforward method where each data moment is preprocessed using a “texture function”, which in effect captures the local *variability* of that moment (noting the observations from Browning [3] that “fluctuations” often have more discriminating utility than the raw values) [17].

- The 2012 paper from T. Islam *et al.* is explicitly an extension of the aforementioned study, but using different AI methods including support vector machine (SVM), decision trees, neural networks, and nearest-neighbor. The simple texture function from Rico-Ramirez was again used in lieu of complicated heuristics.

This paper claimed quite impressive accuracy (98%) as long as dual-pol moments and Doppler velocity are available [8].

- In 2015-2016, a series of papers from Hansoo Lee *et al.* explored a wide variety of AI techniques, including neural networks, discrete wavelet transforms, decision tree ensembles, and support vector machines – but all based on the idea of “spatial clustering” as a pre-processing step. Apparently the idea is that you divide the image into clusters just using the reflectivity data, and then compute statistical information over the whole cluster before feeding

it into the relevant AI system. Although interesting, it is clear that these techniques would be heavily dependent on the effectiveness of the clustering heuristic [12] [11] [15] [13].

- But then in 2018, Hansoo Lee *et al.* offered something different: a classifier using a 3D convolutional neural network (CNN). Finally there would be no need for pre-processing, feature extraction, or clustering – you just feed all the data into the network as a 480x480x41 block of inputs, and let the network decide for itself what local features are applicable [14].

This is a lot more like the approach we took, except that it requires volumetric scanning patterns (which are often not available in the environments we deal with, as will be discussed more in Chapter 4).

- One more article we found interesting was from Li Peng, 2019. Although this one is about identifying targets with military radars rather than the meteorological applications discussed above, it still deals with the same types of issues – namely, the problems of feature selection, model construction, and hyperparameter tuning.

The author notes that these tasks require considerable time and effort from domain experts and data scientists, and so puts forward some ideas based on a concept called “meta-learning”, which allows the user to apply machine learning to figure those parts out instead of doing them by hand.

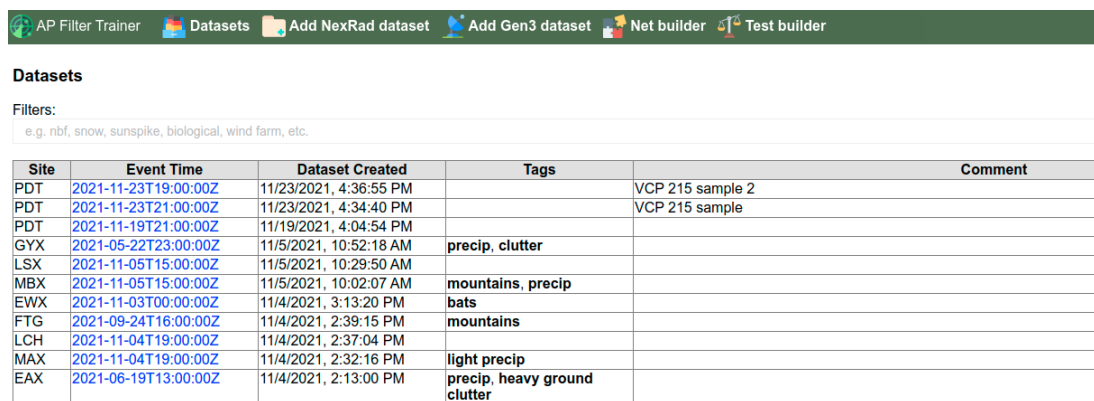
It is an interesting concept, but targeted more for “big data” environments, *i.e.*, when you already have a large library of training data and just need to figure out what to do with all of it.

Chapter 3. ClearScan System Operation

3.1 Sample Import

As with any machine-learning system, providing sufficient training data is key. We provide a streamlined workflow for this process, allowing an operator to quickly select the data to use and label it, and to keep the data samples organized.

The initial view is a list of all data samples that have been imported into the system so far. A navigation bar sits at the top of this view, providing access to other functions which will be described later.



Site	Event Time	Dataset Created	Tags	Comment
PDT	2021-11-23T19:00:00Z	11/23/2021, 4:36:55 PM		VCP 215 sample 2
PDT	2021-11-23T21:00:00Z	11/23/2021, 4:34:40 PM		VCP 215 sample
PDT	2021-11-19T21:00:00Z	11/19/2021, 4:04:54 PM		
GYX	2021-05-22T23:00:00Z	11/5/2021, 10:52:18 AM	precip, clutter	
LSX	2021-11-05T15:00:00Z	11/5/2021, 10:29:50 AM		
MBX	2021-11-05T15:00:00Z	11/5/2021, 10:02:07 AM	mountains, precip	
EWX	2021-11-03T00:00:00Z	11/4/2021, 3:13:20 PM	bats	
FTG	2021-09-24T16:00:00Z	11/4/2021, 2:39:15 PM	mountains	
LCH	2021-11-04T19:00:00Z	11/4/2021, 2:37:04 PM		
MAX	2021-11-04T19:00:00Z	11/4/2021, 2:32:16 PM	light precip	
EAX	2021-06-19T13:00:00Z	11/4/2021, 2:13:00 PM	precip, heavy ground clutter	

Figure 3.1: Datasets with navbar.

But before a dataset will show up here, it needs to be imported; so we will examine that process. Selecting “Add NexRad Dataset” from the navbar brings us to this page, see figure 3.2.

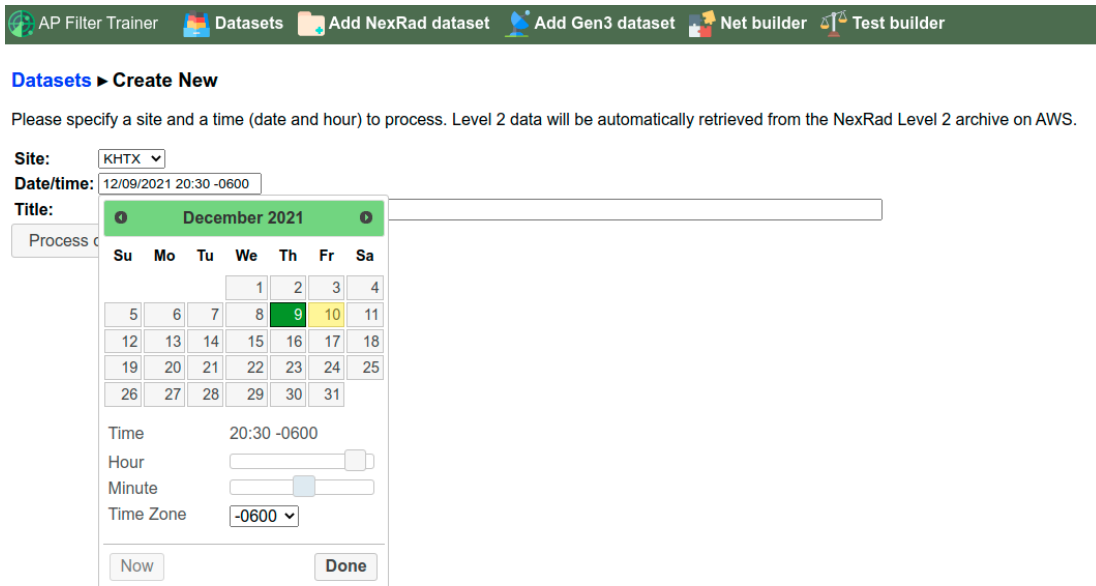


Figure 3.2: Dataset import menu.

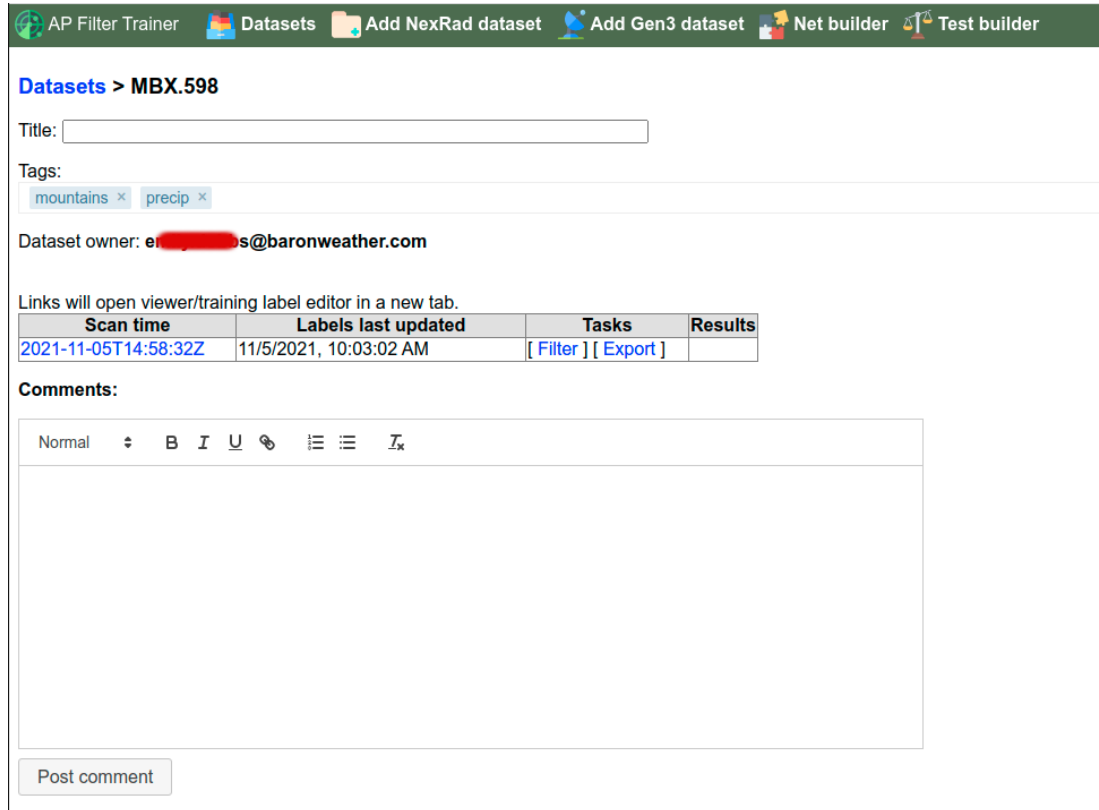
Using this page, it is possible to import any dataset from a NEXRAD site, from a public archive that goes back several decades: <https://registry.opendata.aws/noaa-nexrad/>

The user need only specify a radar site and a date/time, and the system will automatically import that data into the local database for labeling/training purposes.

The “Add Gen3 Dataset” function is similar, but instead provides access to a private archive for custom radars – more on that in Chapter 5.

3.2 Labeling

Once the desired data is imported, you return to the Datasets view and select it. Datasets can include multiple scans *i.e.*, from different vertical tilts (but for now we have it configured to only work with the lowest tilt).



AP Filter Trainer Datasets Add NexRad dataset Add Gen3 dataset Net builder Test builder

Datasets > MBX.598

Title:

Tags: mountains × precip ×

Dataset owner: er...@baronweather.com

Links will open viewer/training label editor in a new tab.

Scan time	Labels last updated	Tasks	Results
2021-11-05T14:58:32Z	11/5/2021, 10:03:02 AM	[Filter] [Export]	

Comments:





Normal **B** *I* U    

Figure 3.3: Dataset page.

Selecting the scan then brings up the labeling interface, see figure 3.4.

A key point for the labeling interface is that it is not necessary to label *every* part of the imported radar image. A user can label as much or as little as they choose, and only the labeled pixels will participate in training, validation,

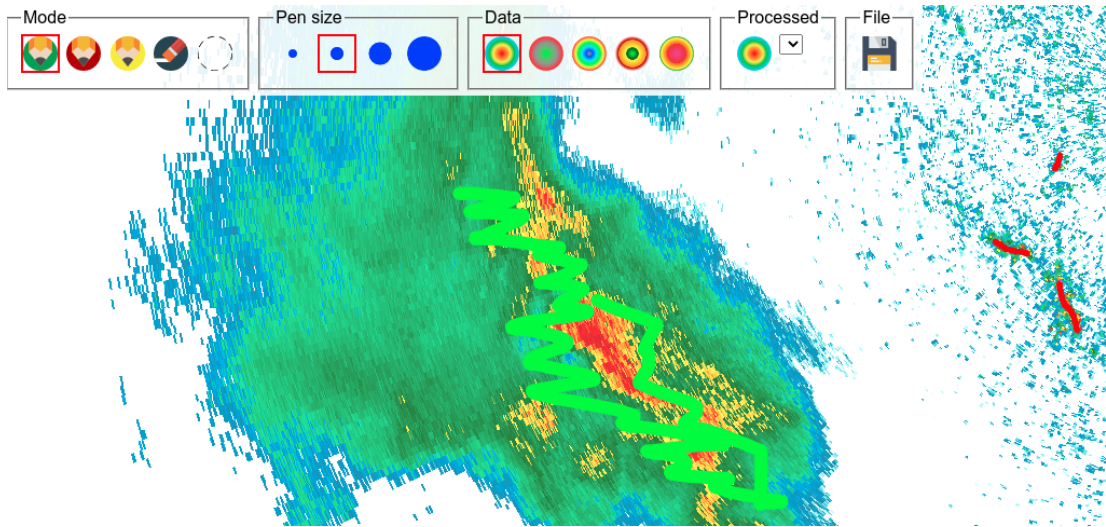


Figure 3.4: Labeling interface.

or testing as applicable. In figure 3.4, the green squiggly line on the left is where the user labeled examples of “good” images, and the smaller red lines to the right are “bad” areas.

Again, these labels are simply drawn onto the image, as if with a paint program. The controls in the upper left allow you to select a “pen color”, with red representing AP (to be removed) and green representing precip (to be kept). There is also an eraser, and an option to hide the labels.

Additional controls along the top include:

- pen size selection;
- data selector, which lets you see other data moments such as velocity and the dual-pol moments, or filter results if present (more on that in a bit);
- a save button.

3.3 Network Builder

Once you have a sufficient number of training samples, it is time to use them to build a filter network. We have found that labeling about 10-12 radar images is sufficient; data augmentation techniques such as “random cropping” are used to help ensure that we do not need to provide enormous quantities of training data (more on that in Chapter 4).

The Network Builder interface (see figure 3.5) lets you specify a name for your network, and select which samples to use for training, validation, and testing. You can also tune some meta-parameters.

The screenshot shows the 'Network builder > Edit Network' interface. At the top, there is a navigation bar with icons for 'AP Filter Trainer', 'Datasets', 'Add NexRad dataset', 'Add Gen3 dataset', 'Net builder', and 'Test builder'. Below the navigation bar, the 'Name' field is set to 'amalthea' and the 'Style' is set to 'Kappa (CNN)'. Under the 'Tuning' section, 'Layer 1 filters' is set to 28 and 'Layer 2 filters' is set to 15. Below these settings, there is a section for selecting datasets for training, with a note: 'Select datasets for training. All datasets with training labels are shown. (Click on timestamp links to review training labels in a new tab.)'. A 'Filters' input field contains the text 'e.g. nbf, snow, sunspike, biological, wind farm, etc.'. Below the filters, there are 'Select all' and 'Deselect all' buttons. The main part of the interface is a table with columns for 'Train', 'Validate', 'Test', and 'Scan time'. The table lists several datasets with their respective tags and scan times, and checkboxes for selecting them for training, validation, or testing.

Train	Validate	Test	Scan time
HTX: [tags: precip]			
<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	2021-10-30T16:56:30Z
HTX: [tags: ground clutter]			
<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	2021-11-01T20:45:11Z
FTG: [tags: light precip, second trip echo]			
<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	2021-11-01T08:25:11Z
RIW: [tags:]			
<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	2021-11-02T15:25:48Z
LZK: [tags: light precip, ground clutter]			
<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	2021-11-02T15:33:57Z
ESX: [tags:]			
<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	2021-11-02T15:43:20Z

Figure 3.5: Network editor.

On the Network Builder overview page (figure 3.6), you can see networks that have been added. Note that we have not trained our new network (“Amalthea”) yet, so no Test Score appears.

Actions	Name	Test score
[edit] [copy] [delete] [train]	amalthea	--
[edit] [copy] [delete] [train]	13_copy6	90.7%
[edit] [copy] [delete] [train]	13_copy4 (copy)	--
[edit] [copy] [delete] [train]	13_copy4	87.0%
[edit] [copy] [delete] [train]	13_copy3	87.7%
[edit] [copy] [delete] [train]	13_copy2	86.8%
[edit] [copy] [delete] [train]	13_copy	88.4%

Figure 3.6: Network builder overview.

We will select “train” to see how it does with the samples we have provided. The training view provides scrolling text that updates us in real-time of the training progress, as shown in figure 3.7.

```

[24] Cost: 0.65124
[25] Preparing sample data
[25] Running batches
[25] Cost: 0.64865
[26] Preparing sample data
[26] Running batches
[26] Cost: 0.64328
[27] Preparing sample data
[27] Running batches
[27] Cost: 0.63920
***** [validate] Cost: 0.86706 Acc: 13.29%
[28] Preparing sample data
[28] Running batches
[28] Cost: 0.63615
[29] Preparing sample data
[29] Running batches
[29] Cost: 0.63291
[30] Preparing sample data
[30] Running batches
[30] Cost: 0.62943
[31] Preparing sample data
[31] Running batches
[31] Cost: 0.62693
***** [validate] Cost: 0.81444 Acc: 18.56%
[32] Preparing sample data
[32] Running batches

```

Figure 3.7: Network training.

Every four batches it will run the samples we selected as “validation” to demonstrate progress. Note that these “validation” samples are not actually used for the training, *i.e.*, it will not update any parameters during that phase; this is mainly to help you see if “overtraining” may be taking place.

At any time we can click “Stop” and conclude the network training. At that point it loads the samples we have selected as “test”, and produces a final score (see figure 3.8).

```

[83] Preparing sample data
[83] Running batches
[83] Cost: 0.47723
***** [validate] Cost: 0.17503 Acc: 82.50%
[84] Preparing sample data
[84] Running batches
Sending stop command
[84] Cost: 0.47729
Finished Training
Reading test samples
IND.562/2021-08-12T22:55:35Z@1.0: AP 11892, PRECIP 17582
BMX.566/2021-08-29T00:58:54Z@1.0: AP 38433, PRECIP 11699
HTX.558/2021-11-02T10:56:07Z@1.0: AP 35156, PRECIP 2948
AMA.569/2021-11-03T14:24:00Z@1.0: AP 5168, PRECIP 0
CBW.572/2021-07-21T09:59:18Z@1.0: AP 0, PRECIP 22985
MAX.588/2021-11-04T19:22:52Z@1.0: AP 0, PRECIP 5438
+++ [TEST] Cost: 0.10888 Acc: 89.11%
--- TASK COMPLETE ---
=== Return to Network Builder ===
--- Connection closed ---

```

Figure 3.8: Network training completed.

Returning to the Network Builder page, we can now see that it has filled in the “Test Score”, as shown in figure 3.6.

But this leaves an important question. Is this new network *really* performing well compared to the other networks... or did we just have different samples selected as the “test” for those other networks? To do a proper comparison, then, we need to define a single test that can be applied to multiple networks.

Actions	Name	Test score
[edit] [copy] [delete] [train]	amalthea	89.1%
[edit] [copy] [delete] [train]	13_copy6	90.7%
[edit] [copy] [delete] [train]	13_copy4 (copy)	--
[edit] [copy] [delete] [train]	13_copy4	87.0%
[edit] [copy] [delete] [train]	13_copy3	87.7%
[edit] [copy] [delete] [train]	13_copy2	86.8%

Figure 3.9: New network added.

3.4 Test Builder

This brings us to the Test Builder page, illustrated in the figures below.

- Figure 3.10 depicts the overview page that shows all the tests that have been defined (it only has one at the moment).
- Figure 3.11 shows the Edit page, where we select the samples we want to use in this test (much like the Edit Network page, except that there is only one checkbox column here to either use the sample or not).
- There is also a “Run” page where we can select a network and run the test, see figure 3.12.

Actions	Name
[edit] [copy] [run]	Full Test

[Create new test](#)

Figure 3.10: Test builder overview.

Test Builder > Edit Test

Name: Favorite

Select datasets for test. All datasets with training labels are shown.

(Click on timestamp links to review training labels in a new tab.)

Filters:

Include	Scan time
HTX: [tags: precip]	
<input type="checkbox"/>	2021-10-30T16:56:30Z
HTX: [tags: ground clutter]	
<input type="checkbox"/>	2021-11-01T20:45:11Z
FTG: [tags: light precip, second trip echo]	
<input type="checkbox"/>	2021-11-01T08:25:11Z
RIW: [tags:]	
<input type="checkbox"/>	2021-11-02T15:25:48Z
LZK: [tags: light precip, ground clutter]	
<input type="checkbox"/>	2021-11-02T15:33:57Z

Figure 3.11: Editing the list of test samples.

If we select our test as a “Favorite”, a column for its results will be shown in the “Network Builder” overview page as shown in figure 3.13.

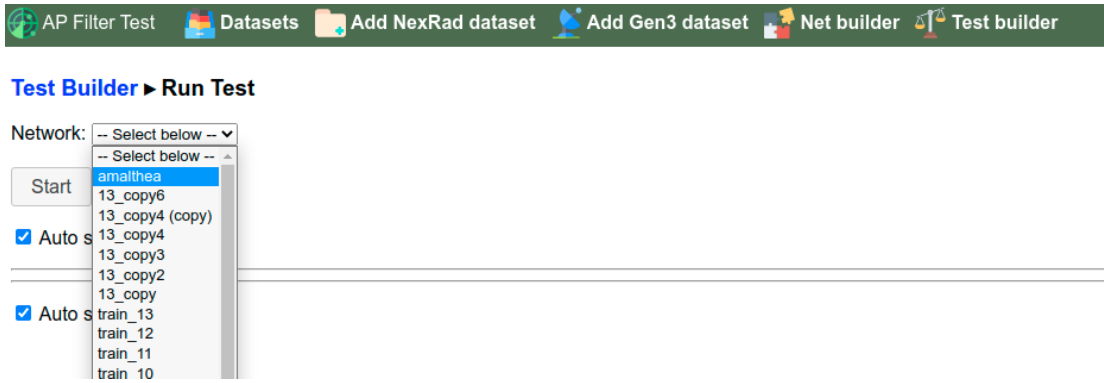


Figure 3.12: Selecting a test to run.

The screenshot shows the 'Network builder' interface with a table of test results. The table has four columns: 'Actions', 'Name', 'Test score', and 'Full Test'. The 'Full Test' column is highlighted in blue. The table contains the following data:

Actions	Name	Test score	Full Test
[edit] [copy] [delete] [train]	amalthea	89.1%	89.2%
[edit] [copy] [delete] [train]	13_copy6	90.7%	87.8%
[edit] [copy] [delete] [train]	13_copy4 (copy)	--	--
[edit] [copy] [delete] [train]	13_copy4	87.0%	89.3%
[edit] [copy] [delete] [train]	13_copy3	87.7%	88.3%
[edit] [copy] [delete] [train]	13_copy2	86.8%	--
[edit] [copy] [delete] [train]	13_copy	88.4%	--

Figure 3.13: Results column for Full Test shown.

Chapter 4. Implementation Details

As we can see from Chapter 2, there are many approaches that people have tried over the years to solve this problem, often achieving impressive accuracy.

Although it might be tempting to follow the most successful methods like a recipe, the reality is that our operational environments are not necessarily the same. In particular, we do not always have volumetric data, as some radar users prefer to only scan at a single elevation; therefore, the 3D CNN as in H. Lee 2018 [14] would not quite work. And the various methods involving pre-processing are tuned for specific types of image contamination from AP, and are not necessarily applicable for other problems such as RF interference (as will be discussed more in the next chapter).

As mentioned in the introduction, CNNs offer substantial flexibility by allowing the network to decide for itself what features are important, rather than requiring domain experts to implement and tune robust feature extraction methods. And the hope further is that the use of dual-polarization data will provide sufficient input to the CNN to make up for the lack of volumetric input.

We are not using Doppler velocity data because, as Lakshmanan [10] and others have noted, this would severely restrict the useful range of the network.

(We could train two networks like Lakshmanan did, but we have not opted to do so.)

Consequently, the inputs to our CNN are the raw reflectivity, and the three dual-pol data moments, namely differential reflectivity, differential phase, and correlation coefficient (see [7]).

An image containing these four data moments as “channels” (much like an RGB image would have three channels) is then fed through a CNN, implemented using PyTorch [16], like this:

```
import torch
import torch.nn as nn

class KappaNet(nn.Module):
    def __init__(self, layer_1_filters=28, layer_2_filters=15):
        super(KappaNet, self).__init__()
        self.conv1 = nn.Conv2d(4, layer_1_filters, (5, 5))
        self.conv2 = nn.Conv2d(layer_1_filters, layer_2_filters, (3, 3))
        self.conv3 = nn.Conv2d(layer_2_filters, 1, (3, 3))

    def forward(self, x):
        x = torch.tanh(self.conv1(x))
        x = torch.tanh(self.conv2(x))
        x = torch.tanh(self.conv3(x))
```

```
return x
```

Note, there are no fully-connected layers here, so if you give it, say, a 32x32 image, then the output will be a 24x24 image. The intent is to train it such that each output pixel is a classification value, +1 for AP/non-precip or -1 for precipitation.

No “padding” is used here explicitly, because we only want to classify pixels for which the full 9x9 “context” input region is available. (In practice, if we have a full 360 degree scan, then we can just replicate radials from the end of the scan to the beginning, in order to provide the needed context so as not to leave those radials unclassified.)

When training, remember that not every pixel is labeled (see Chapter 3), so it is important that the “loss” function only be influenced by results for actually labelled pixels. So the training loop looks something like this:

```
criterion = nn.MSELoss(reduction='sum')
sgd = functools.partial(optim.SGD, lr=0.0002, momentum=0.5, weight_decay=0.01)
optimizer = sgd(net.parameters())
for inputs, labels in sample_batches(trainval, shuffle=True):
    optimizer.zero_grad()

    outputs = net(inputs)
    mask = torch.abs(labels)
    n = mask.count_nonzero()
    loss = criterion(outputs * mask, labels) / n
```

```
loss.backward()
optimizer.step()

running_loss += loss.item() * int(n)
```

Here, “labels” is an array with -1.0 for pixels labeled as precip, +1 for non-precip, and 0 for unlabeled pixels. So `torch.abs` results in just 1 for labelled and 0 for unlabelled. Multiplying this by the `outputs` array then treats it as if the network answered 0 for the unlabelled pixels as well, so that the `MSELoss` function will not count those as contributing to the loss.

As far as generating training sample images, basically any 32x32 image centered on a labelled pixel can be selected. This is comparable to the well-known data augmentation technique of random cropping (see, for example, [18]), which makes training feasible with only a moderate number of labelled images. The routine goes something like this:

```
def get_samples(n, shuffle=True):
    index = numpy.argwhere(train_labels > 0)
    count = index.shape[0]
    if shuffle:
        candidates = numpy.arange(0, count)
        numpy.random.shuffle(candidates)
        sel = candidates[:n]
    else:
        sel = numpy.arange(0, n)
```

```

boxes = numpy.empty(shape=(n, 4, 32, 32), dtype=numpy.uint8)
targets = numpy.empty(shape=(n, 1, 24, 24), dtype=numpy.uint8)
for i, j in enumerate(sel):
    y, x = index[j, :]
    boxes[i, :, :, :] = train_data[:, y - 16:y + 16, x - 16:x + 16]
    targets[i, 0, :, :] = train_labels[y - 12:y + 12, x - 12:x + 12]
return boxes, targets

```

The above provides randomly selected 32x32x4 training images (or not randomly, in case we're just running the validation set), as well as the 24x24x1 "target" images (-1 for precip and +1 for non-precip).

There is a lot more that could be said about the supporting code, but the above is really the core of the machine learning techniques applied.

Chapter 5. Case Study: Bangladesh Radar

5.1 Introduction

Baron Weather was contracted to build two radars in Bangladesh. Once they were built and operational, we found that there was an extreme amount of RF interference in the area which was heavily contaminating the images, see figure 5.1.

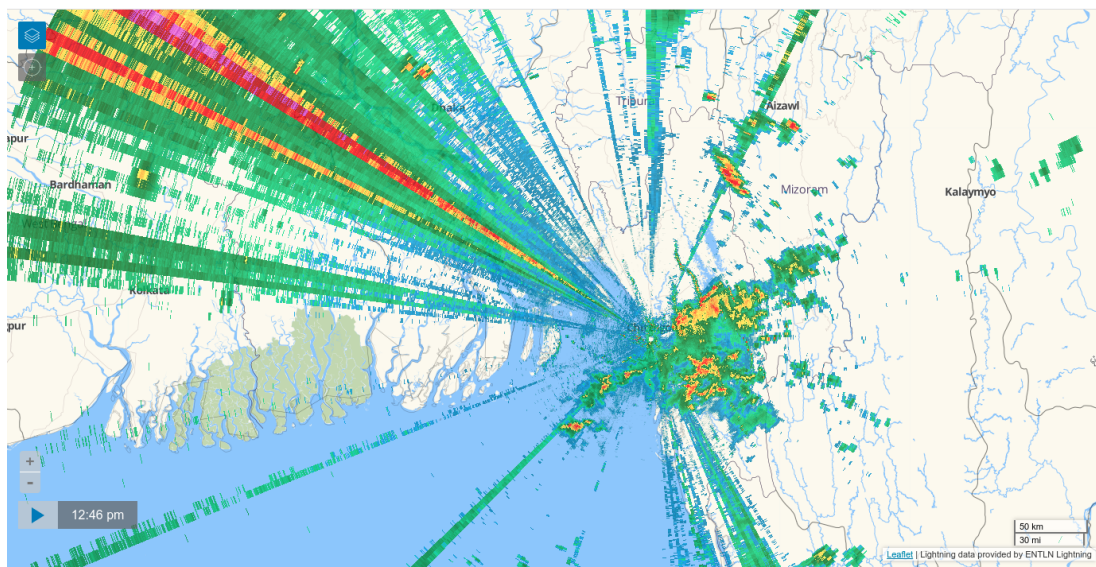


Figure 5.1: Bangladesh radar with contamination.

Our understanding is that this happens because RF frequencies are not as closely regulated there as they are in the United States. But whatever the reason, the users were quite unhappy with the data they were getting.

As a matter of course, we tried standard tools such as CLEAN-APTM [1]; but those were not sufficient. It seems likely that the severe image contamination here is just not the sort of use case that off-the-shelf tools are designed for. What we needed was a filter that was trained *specifically for these operating conditions*, using *data from these radars*.

5.2 Dual Networks

When working with the Bangladesh radar data, we decided early on that we should separate samples of RF interference from samples of “plain” AP. This would allow us to train two separate networks, each being trained only with one of the two categories of image contamination.

So, when importing the sample data, we used tags to keep track of which type of AP was being labeled. Figure 5.2 shows a sample with just the RF areas labeled. Figure 5.3 shows another sample where just the “plain” AP areas are labeled.

Note that in both cases, areas of “good” precip are still labeled, since we want the network to learn to distinguish the two.

Other options considered were to just lump all the “bad” areas together, or to train a single network that could output three possible classes (precip, plain

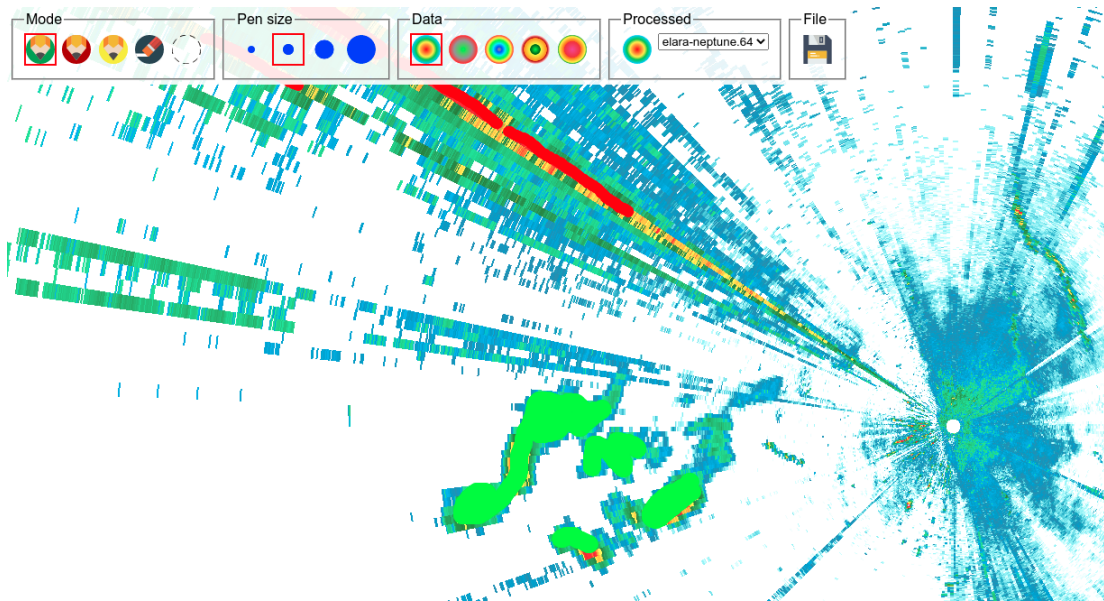


Figure 5.2: RF labeled image.

AP, and RF interference). Due to limited time available, though, the decision was made to stick with the separate networks plan.

5.3 Results

We trained two networks as described, then built a two-layer filter. Echoes that were rated as “bad” by either network would be removed. The resulting images were significantly cleaner, see figure 5.4.

5.4 Alternative Network Comparison

After seeing these results, a question arose as to whether the complexity of a CNN is really needed. After all, [10] managed to create a successful filter using a simple FCN (though, again, relying heavily on pre-processing).

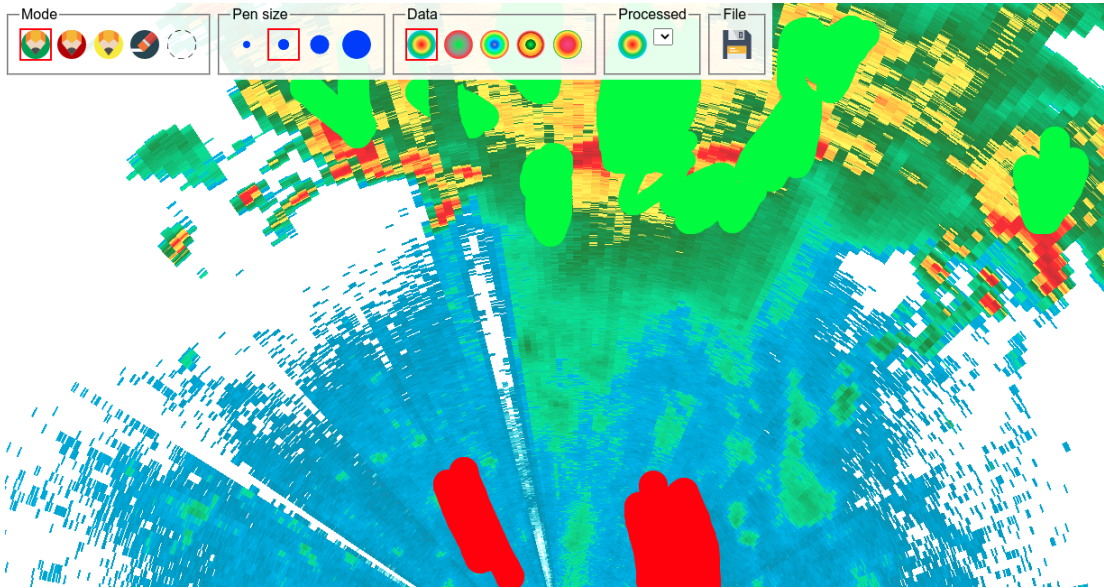


Figure 5.3: Plain AP labeled image.

For comparison purposes, we added an alternative type of network that can be selected when training a network. This new network style is a FCN which takes the reflectivity and dual-pol moments as four scalar inputs for a single pixel, and produces an answer for that pixel. Note that no local-neighborhood data is involved for this type – it just tries to come up with an answer strictly using the data for the point in question.

The results were not as good – see figure 5.5.

Again, this network structure is only “inspired” by [10], and is not represented as being a reproduction of that work – particularly since we did not implement any of the pre-processing described in that paper. This was just done as a sanity check, to see if the complexity of our original solution was merited.

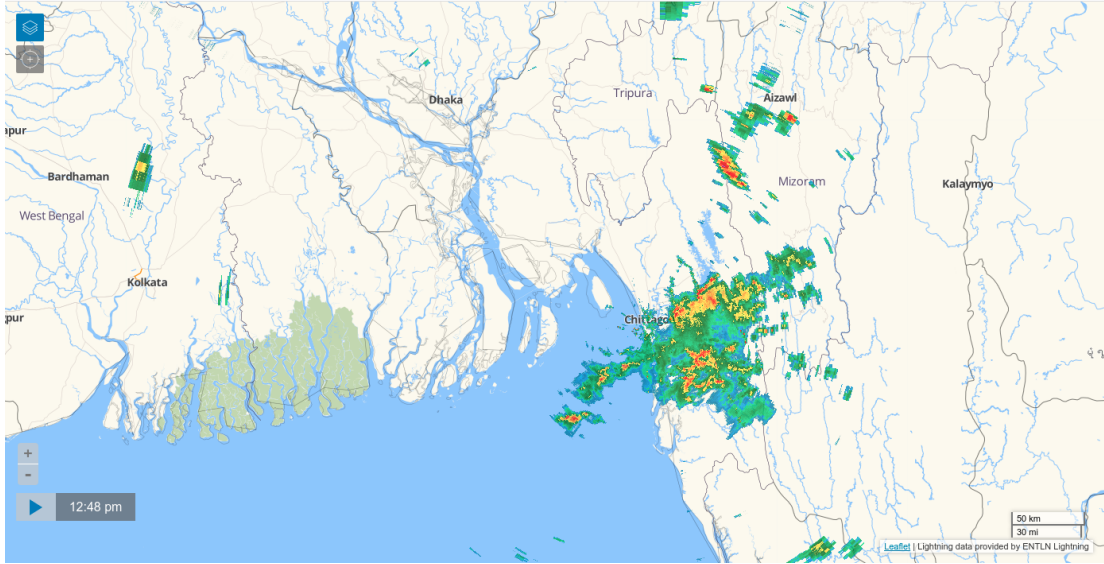


Figure 5.4: Bangladesh radar after cleanup.

5.5 Quantitative Results

The pictures shown here say a lot about the results; and indeed, on the strength of demo images like these, the decision was made to ship this as a production system for the Bangladesh radar. We are even seeking a patent, which, as of this writing, has been filed but approval is still pending.

However, from a scientific standpoint, it is sometimes nice to have quantitative measurements of how successful the system is. So using the Test Builder (see section 2.4), we measured the performance for the networks trained in for this case study. The results were as shown in figure 5.6.

Since each network was trained on only RF contamination or only plain AP, then it also made sense for the tests to be likewise segregated, to see how well the network performs just at what it was intended to do.

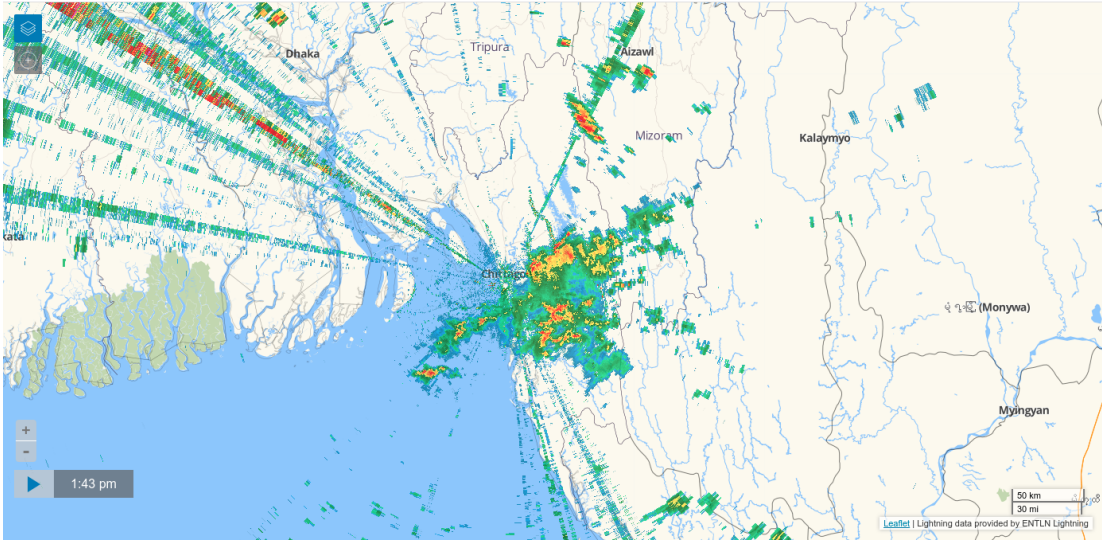


Figure 5.5: Alternative filter.

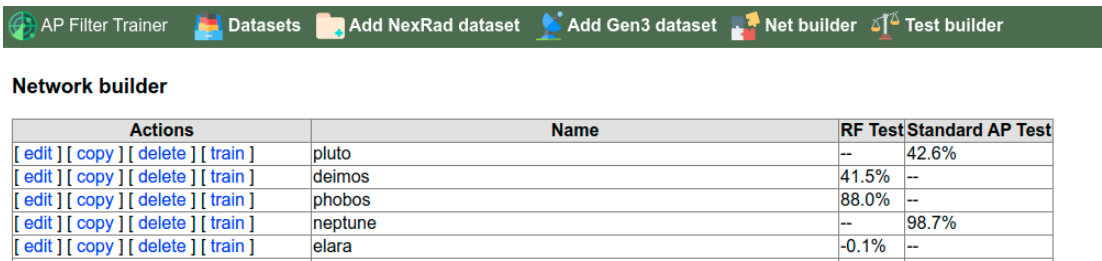


Figure 5.6: Bangladesh filter test scores.

Networks “elara”, “phobos” and “deimos” in this table were trained for RF interference, so I ran the RF test against those. Likewise, networks “neptune” and “pluto” were trained for plain AP.

As described in the previous section, networks “pluto” and “deimos” were built using the alternative, simplified network structure. Although these networks did quite well with the training samples (with scores over 90 percent), the test shows that they do not perform that well. This agrees with what we observed

in the visual results; and suggests that the initial decision to focus on CNNs was likely appropriate.

A note on score formula: In an early version of this project, we used a simple “accuracy” measure, which takes the network’s answers for each point in the test and just turns it into a binary decision based on which side the response is closest to. The accuracy, then, was a measure of how many points are answered correctly in this manner.

However, such a measurement ignores how strong the response is, and would give the same score to a network providing answers that were “lukewarm” but correct when considered as binary decisions, as it would a network giving “strong” responses.

The current formula is based on a mean-squared-error metric, where the network is expected to answer +1.0 for “bad” pixels or -1.0 for “good” pixels. This formula would yield 0.0 for perfect results, 4.0 for worst-case, and 1.0 for all-neutral results (*i.e.*, if the network always answered 0).

The final “accuracy” score is taken as $(1.0 - \text{MSE}) * 100$. This puts the all-neutral case at 0 percent, and 100 percent would be a perfect score. Negative scores, then, are possible (as you can see from the table above – network “elara” was trained using a very minimal set of data, so it did not perform very well).

5.6 Potential Drawback

It is perhaps the case that we have made this system out to be ideal; but in practice we have encountered difficulty from time to time. In this section we will

describe an issue we encountered operationally that underscores one weakness of this approach to solving the problem.

After we had trained the network for Bangladesh and had been running it for a while, we received a report that it was no longer doing a good job. Indeed, the filter was wiping out *all* of the precip.

Figure 5.7 shows the un-filtered radar returns; and after filtering it looked as shown in figure 5.8.

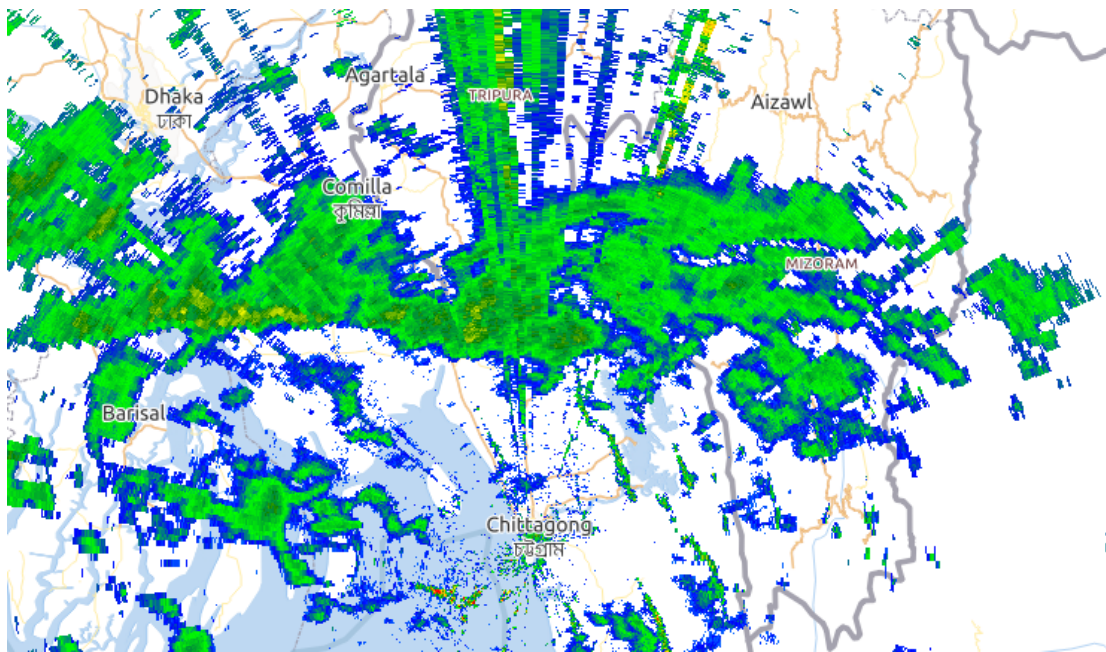


Figure 5.7: Bad precip case.

Upon investigating, we learned that a hardware component had been replaced which changed the physical length of the receiver, and this had caused a significant change to the differential phase dual-pol moment. Here is what that data looked like:

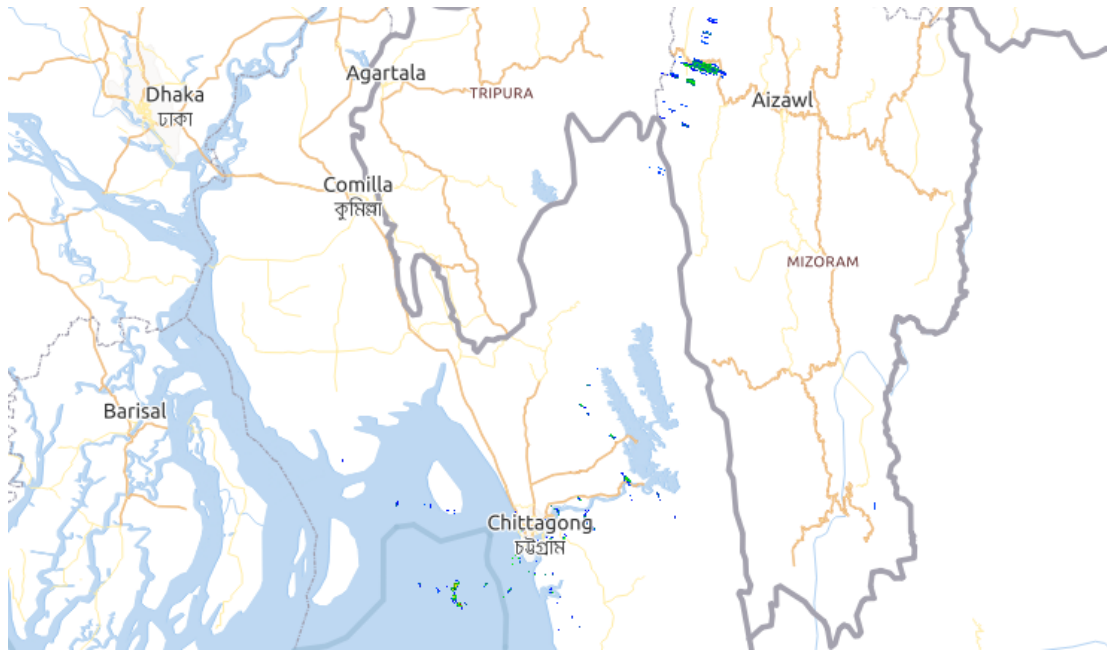


Figure 5.8: Bad filter result.

Note that in areas that should be valid precip, the data is all one color (likely because it was clipping beyond the valid range). Figure 5.10 is an example from another day which shows what differential phase is supposed to look like.

We believe what we are seeing here illustrates a weakness that CNNs and many other machine-learning systems have, which is that they come to expect the data to be like what they were trained with; so when a major shift or bias is introduced into the input then recognition capability is greatly diminished. Even the use of a separate test dataset to check for overtraining fails to discover the weakness, because the images in the test dataset, like the training dataset, all predated the hardware change that introduced the bias.

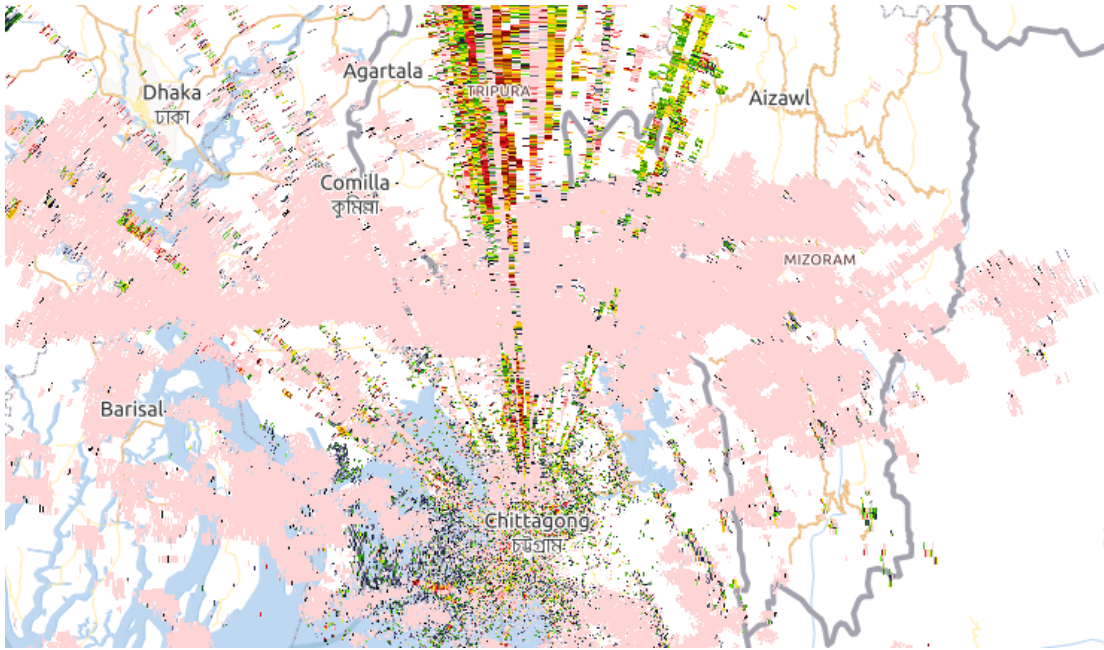


Figure 5.9: Bad PHI (Differential Phase) data.

It is a known issue that has received some research attention. For example, Bahng *et al.* refers to this as the “cross-bias generalization problem”, and comments: “A model that relies on bias will achieve high in-distribution accuracy, yet fail to generalise when the bias shifts” [2].

Certainly, one solution available to us would have been to train a new network using only data from after the shift. As it turns out, though, there was a recalibration procedure available for the radar which restored the differential phase data back to the way it was; and the original filter started working normally.

This did spur some discussion about how to make this more robust in case the radar “slips out of calibration” over time. As noted in Bahng *et al.*, a number of data augmentation techniques are known that could help the network

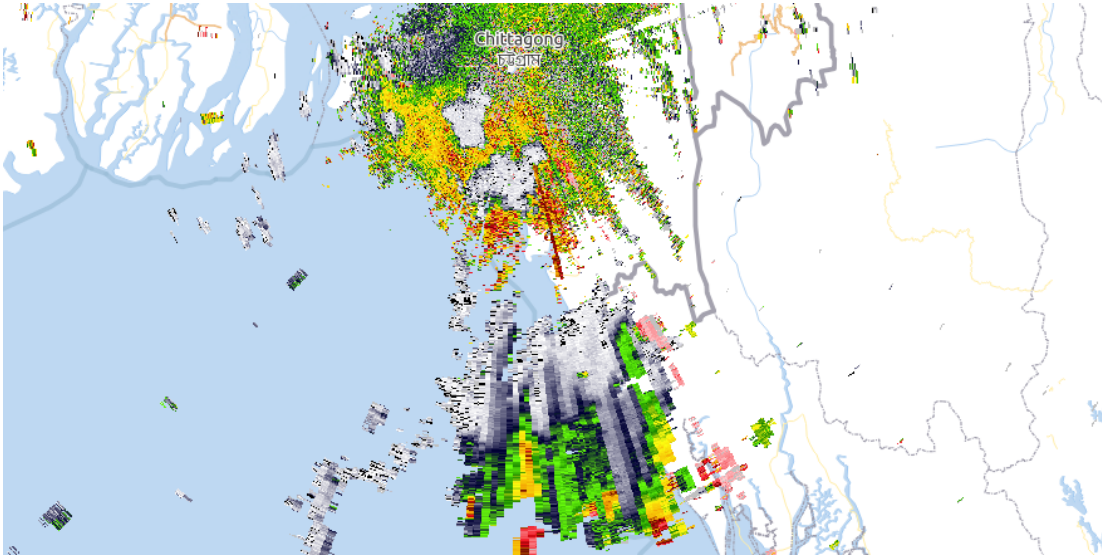


Figure 5.10: Good PHI (Differential Phase) data.

generalize better [2]. It may also be possible to pre-process the data in a way that this bias is not visible. For example, we could use Specific Differential Phase (KDP) instead of raw differential phase [19]; or even something like the “texture function” from Rico-Ramirez [17]. This remains an area for us to experiment.

Chapter 6. Conclusion

In this section we will summarize the contribution of this research and discuss potential areas of future work.

6.1 Contribution

The ClearScan system provides a robust machine-learning environment with an intuitive interface, enabling the creation of weather radar image filters that are customized to the particular environment and operational characteristics of a particular radar.

This has proved particularly important given the varying operational environments we have encountered with radar installations. RF interference in particular seems to have different characteristics in different areas, which would make any one-size-fits-all image filter unlikely to succeed. Experience has shown that the network needs to be trained on data from that radar for optimal effectiveness; and having a streamlined process for this has been invaluable.

6.2 Future Work

- As mentioned at the end of Chapter 5, there is still a concern that a trained filter could become less effective over time, either because the radar drifts

out of calibration or because the RF interference becomes drastically different; nevertheless, we have not been using this system long enough to know how prevalent these problems will be. Certainly there is a desire to improve the robustness of the trained networks to cut down on the necessity of re-training filter networks after they're deployed.

- There is also still a demand for running a system like this without the use of dual-pol moments, *i.e.*, just using reflectivity data. With the system as is, we have found that it cannot learn the patterns effectively just from that data. It may be worthwhile to try using successive scan information to detect “movement” as described by Grecu [6], to see if this can improve the learning effectiveness for a reflectivity-only case.

References

- [1] Clean-ap™. <https://web.archive.org/web/20201004094649/https://www.baronweather.com/clean-ap/>.
- [2] Hyojin Bahng, Sanghyuk Chun, Sangdoon Yun, Jaegul Choo, and Seong Joon Oh. Learning de-biased representations with biased representations. *arXiv.org*, 2020.
- [3] K A Browning. Meteorological applications of radar. *Reports on progress in physics*, 41(5):761–806, 1978.
- [4] V. Chandrasekar, R. Keränen, S. Lim, and D. Moisseev. Recent advances in classification of observations from dual polarization weather radars. *Atmospheric Research*, 119:97–111, 2013. ADVANCES IN PRECIPITATION SCIENCE.
- [5] Grecu, Mircea, and Witold F Krajewski. An efficient methodology for detection of anomalous propagation echoes in radar reflectivity data using neural networks. *Journal of atmospheric and oceanic technology*, 17(2):121–129, 2000.
- [6] M. Grecu and W.F. Krajewski. Detection of anomalous propagation echoes in weather radar data using neural networks. *IEEE transactions on geoscience and remote sensing*, 37(1):287–296, 1999.
- [7] Paul H. Herzegh and Arthur R. Jameson. Observing precipitation through dual-polarization radar measurements. *Bulletin of the American Meteorological Society*, 73(9):1365 – 1376, 1992.
- [8] Tanvir Islam, Miguel A Rico-Ramirez, Dawei Han, and Prashant K Srivastava. Artificial intelligence techniques for clutter identification with polarimetric radar signatures. *Atmospheric research*, 109-110:95–113, 2012.
- [9] G. N. Johnson, P. L. Smith, F. E. Nathanson, and L. W. Brooks. An analysis of techniques for dealing with anomalous propagation., 1975.

- [10] Valliappa Lakshmanan, Angela Fritz, Travis Smith, Kurt Hondl, and Gregory Stumpf. An automated technique to quality control radar reflectivity data. *Journal of Applied Meteorology and Climatology*, 46(3):288 – 305, 2007.
- [11] H. Lee, E.K. Kim, and S. Kim. Anomalous propagation echo detection using neural network and discrete wavelet transform. In *Proceedings of the 2015 International Conference on Artificial Intelligence and Industrial Engineering*, pages 188–190. Atlantis Press, 2015/07.
- [12] Hansoo Lee, Eun Kyeong Kim, and Sungshin Kim. Anomalous propagation echo detection using artificial neural network and doppler velocity features. In *2015 IEEE International Conference on Advanced Intelligent Mechatronics (AIM)*, pages 377–381. IEEE, 2015.
- [13] Hansoo Lee, Eun Kyeong Kim, and Sungshin Kim. Anomalous propagation echo classification of imbalanced radar data with support vector machine. *Advances in meteorology*, 2016:1–13, 2016.
- [14] Hansoo Lee, Jonggeun Kim, and Sungshin Kim. A 3d convolutional neural network for anomalous propagation identification. *INTELLI 2018 : The Seventh International Conference on Intelligent Systems and Applications*, 2018.
- [15] Hansoo Lee and Sungshin Kim. Decision tree ensemble classifiers for anomalous propagation echo detection. In *2016 Joint 8th International Conference on Soft Computing and Intelligent Systems (SCIS) and 17th International Symposium on Advanced Intelligent Systems (ISIS)*, pages 391–396. IEEE, 2016.
- [16] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

- [17] M.A Rico-Ramirez and I.D Cluckie. Classification of ground clutter and anomalous propagation using dual-polarization weather radar. *IEEE transactions on geoscience and remote sensing*, 46(7):1892–1904, 2008.
- [18] Ryo Takahashi, Takashi Matsubara, and Kuniaki Uehara. Data augmentation using random image cropping and patching for deep cnns. *IEEE Transactions on Circuits and Systems for Video Technology*, 30(9):2917–2931, 2020.
- [19] Yanting Wang and V Chandrasekar. Algorithm for estimation of the specific differential phase. *Journal of atmospheric and oceanic technology*, 26(12):2565–2578, 2009.