UNIVERSITY OF OKLAHOMA

GRADUATE COLLEGE

DEVELOPMENT OF DEEP LEARNING METHODOLOGIES FOR MODELING NAVIGATIONAL FEATURES IN CONTINUOUS SPACES

A THESIS

SUBMITTED TO THE GRADUATE FACULTY

in partial fulfillment of the requirements for the

Degree of

MASTER OF SCIENCE

By

ZACKARY HOYT
Norman, Oklahoma
2021

DEVELOPMENT OF DEEP LEARNING METHODOLOGIES FOR
MODELING NAVIGATIONAL FEATURES IN CONTINUOUS SPACES

A THESIS APPROVED FOR THE
GALLOGLY COLLEGE OF ENGINEERING

BY THE COMMITTEE CONSISTING OF

To my family and friends,

who supported me throughout this journey

with their kindness, encouragement, and help.

Abstract

This thesis proposes generalized methodologies to model navigational features of continuous spaces using deep learning architectural approaches. Navigational features impact how an entity can effectively travel within a space, such as environmental hazards in the space. Deep learning is a subset of machine learning that utilizes artificial neural networks to effectively solve stochastic optimization problems. Modeled navigational features can be used in conjunction with deep learning architectures to solve stochastic optimization problems in the domain of navigation, such as developing navigational solutions that detail how to effectively travel from one point to any other reachable point in the navigable space.

Deep learning architectural approaches are explored to generate airspace navigational solutions from modeled airspace navigational features as a means to evaluate the proposed deep learning modeling methodologies. An airspace scenario generator is designed and developed to procedurally generate stochastic scenarios of spatial airspace data. Techniques are developed to deterministically convert these scenarios into corresponding navigational solutions to support a supervised learning approach. To evaluate the predictions of trained models against the actual navigation solutions, new statistical measures are introduced to better measure the accuracies and coherences of the models by accounting for both positional distributions and biases in the predictions. Multiple comparable architectural approaches are provided with detailed implementations that empirically demonstrate this capability successfully.

Index Terms—Artificial Intelligence; Autonomous Agents; Deep Learning; Machine Learning; Continuous Spaces; Autonomous Navigation; Automated Routing

Table of Contents

Figures

Algorithms

Tables

Chapter 1 Introduction

The topic of this thesis is selected with respect to personal involvement in a concerted research effort between the United States Air Force Sustainment Center Engineering and Technical Management Directorate (AFSC/EN) and researchers at the University of Oklahoma. The broad description of this research is the development of the Self-Explaining Decision Architecture (SEDA), which entails machine learning (ML) systems that generate explanations for any produced decision (Stringer, et al., 2021). Of particular relevance in the SEDA research is the problem space of developing an interpretable auto-router: a system capable of determining an aircraft's route. The concept of the thesis was expanded from this core idea of autonomous airspace path-finding to a more generalized conceptual look at how navigation in continuous spaces could be autonomously driven. Thus, there are two main components: the titular methodologies to model elements within a continuous space that impact navigation—navigational features—and how to then shape these modeled navigational features into information used to identify effective paths between any two reachable points in the space—navigational solutions. The remainder of the Introduction chapter will more describe the undertakings of this thesis and the results as an extended abstract, as well as how the content of the thesis is arranged.

1.1 – Extended Abstract

Deep learning (DL) is used to solve stochastic optimization problems by utilizing artificial neural networks to model complex, nonlinear relations between input variables and an output that optimizes the objective function of some stochastic optimization problem (Goodfellow, et al., 2016). *Artificial neural networks* (ANNs) emulate biological neural networks where variable feature inputs—input signals—are modeled across a number of artificial neurons (ANs) through a number of synapses that determine the signal strength between two ANs (ibid). ANs are typically grouped into *neural network layers,* which creates an internal state-based representation of the model inputs (ibid). The input signals of a layer are mapped to a number of output signals that can then either be passed into another neural network layer or used to formulate some solution. The terms *navigational features*, *navigational solutions,* and *navigable spaces* are used throughout this thesis to describe, respectively, spatial features that impact how some entity can navigate within a space, information used to navigate a space, and some space that can be navigated. Thus, to model navigational features using DL architectural approaches, the navigational feature information is detailed in internalized state of some ANN such that subsequent neural network layers can transform the information to solve some objective function. Modeled navigational features can then be used to solve stochastic optimization problems that are a function of navigational features. Such a problem, explored by this thesis to evaluate the feasibility of modeling navigational features in a continuous space, details how to determine an effective route between two points in a navigable space.

Navigational features and solutions and navigable spaces detail the main considerations of the problem space explored, and are characterized to be within a generalized airspace domain with respect to how aircraft should traverse the space. This characterization is directly derived from the motivation, but also provides the problem space with some structure that serves to improve interpretability, as opposed to interpreting abstracted data. Thus, these terms are further contextualized throughout this thesis with respect to those considerations of an airspace. While the problem space used to evaluate them is characterized, the proposed methodologies themselves are designed to be generalizable to other similar problem spaces.

Datasets of navigational features and the navigational solutions are procedurally generated by an airspace scenario generator, which stochastically generates spatial navigational features that detail risks which aircraft should avoid. The distribution of risk across some region in the navigable space, with respect to a navigational feature, is denoted as the *gradient of avoidance*. Composing all the avoidance gradients within a space then details the navigational solution of that space. These navigational solutions are then structured as 2-dimensional heatmaps of avoidances, that can be constructed with variable spatial resolution and can be solved by a path-finding algorithm to find an effective route between any two points within the navigable space. This is a generalizable solution, as both the selection of the path-finding algorithm and how the algorithm interprets avoidances can be made specific an individual problem space.

To artificially constrain the ways to solve this problem, the collective gradients of avoidance are considered to be inseparable in the navigational solution; the entirety of a space's navigational features must be collectively modeled and then transformed to yield the composition of collective avoidance gradients as an output. This is to emulate complex problem spaces with entangled feature relations, where the causality of risks is a sufficiently complex function of multiple navigational features. In such a problem space, it would then be infeasible to decompose the problem into a system of simpler problems, whose corresponding solutions can compose the solution of the original problem.

As the architectural approaches will entail a model that makes a prediction of the navigational solution, statistical analysis of the model predictions with respect to the actual navigation solutions, must then account for the positional relativity of predicted avoidance gradients and biases of the problem space—if the average risk of a point in the navigational solution leans closer to either no risk or maximum risk. However, no such existing measures are known. To address these needs, this thesis introduces new statistical measures to evaluate a prediction's loss and accuracy and the model's coherence: n-Balanced Type I and II Error (NBTE) Loss, Accuracy, and Coherence. Note that the Type Error refers to a continuous generalization of the predicted vs actual values; Type I/II Error is incurred from a prediction greater/less than the actual where the penalty is then distance above/below the actual.

NBTE Loss equalizes the proportional penalties of Type I and II Errors (magnitudes of false positives and negatives) made between a predicted and the actual navigational solution. In doing so, it accounts for both positional errors with respect to the biases of the space. *NBTE Accuracy* is directly derived from the *NBTE Loss*; it then conveys accuracy of a prediction with respect to the balanced penalties of Type I and II Errors. *NBTE Coherence* is then used to measure the quality of predictions made by the models. While the proposed accuracy details performance with respect to the loss, this introduced measure of coherence informs of performance with respect to the expected NBTE Loss and Accuracy of a randomized prediction. Thus, the coherence measures how meaningful the model's predictions are by evaluating it with respect to the NBTE Accuracy measures of both completely random and perfect predictions.

Architectural approaches explored utilize the Convolutional Neural Networks (CNNs) to model the navigational features by convolving a discretized view of the space—a multi-dimensional array—and internally stores the information in a number of output neurons structured as a condense discretized view of a space—feature map. This modeling approach yields a navigational feature map that details the internalized navigational feature representation in the DL architecture. This navigational feature map is transformed into a navigational solution by either the Transposed-Convolutional Neural Networks (TCNNs) (Conv-to-ConvT Architecture) or the Dense Neural Networks (DNNs) (Conv-to-Dense Architecture). TCNNs effectively detail a reverse CNN that allows condensed spatial information to be expanded and DNNs can map each input neuron to a configurable number of output neurons that can be reshaped to satisfy the architectural output requirements.

These simple architectural elements are widely used in DL applications and areas of research, which demonstrates the potency of a DL architecture to combine simple elements into effective modeling approaches. Conv-to-ConvT and Conv-to-Dense architectures provide two comparable architectures, where a specific implementation of the best discovered configuration/layout of both architectures is selected to represent their respective architectural style. These two architectures are detailed and compared, and both are empirically demonstrated to be capable of producing the expected navigational solutions, thus verifying the feasibility of the proposed methodologies. The criteria for success are a multifactored consideration of model NBTE Accuracy and Coherence, the statistical confidence in the similarity of the distribution of values in the navigational solution between the predicted and actual navigational solutions, and if the model predictions can be interpreted as matching the actuals. It is further detailed that the measured accuracy, coherence, and distribution similarity of the representatives for both architectural models are nearly equal.

A dimensional variant of the problem space is also briefly explored, which introduces temporality to the airspace scenarios to simulate temporally dynamic behaviors. A sampled scenario is temporalized and then used to train implemented temporal models to forecast future navigational feature states of the sampled scenario. The ability to forecast future conditions is an important consideration of any navigational system, as it can provide ultimately more effective routes that circumvent future hazards that would otherwise have impeded navigation and may detail costly corrections. The architectural approach used to model the navigational features is the basis of this temporal architectural approach, and thus the Convolution-LSTM NN (Conv-LSTM NN) is used to model the spatial-temporal navigational features. Then, a similar strategy to transform the modeled data into a forecast is also used; a Conv-LSTM-to-Dense architecture is used to evaluate this capability. The empirically demonstrated results for this approach detail an unsuccessful approach, where the models learnt the weighted paths of traveling aircraft rather than how to forecast future positions. However, it is not conclusive yet as to whether or not temporality can be successfully accounted for; this yields a basis for more thorough exploration of the domain. Additionally, the output of this temporal architecture can detail half-temporality, in which the information can be used to add some generalized hazard to those regions to represent the potential risk, over time, of traversing those spaces if accurate forecast models cannot be implemented.

1.2 – Contributions

There are multiple contributions this thesis makes with regards to ML-based navigational systems and statistical analysis:

1. The predominant contribution is the centralized navigational feature modeling methodologies in a DL architecture. While the focus is on navigational features, a broader domain in which this can also be applicable to are generalized spatial features. Additionally, the exploration of the problem space to generate navigational solutions yields several empirically successful architectural approaches that can be the bases for actual

auto-routing systems in the future. Specifically, these DL architectural approaches detail a centralized (non-egocentric) system that constructs a comprehensive navigational solution of a target space with respect to the observed navigational features therein.

2. The introduced NBTE measures also offer a means to better evaluate models that yield multi-dimensional arrays of positionally distributed values with potential bias in the expected values of the space; NBTE Loss can be used as a regression loss function to train models, where NBTE Accuracy and Coherence can then be used to evaluate the performance of the predictions and models, respectively.
3. This provision of specifications for developing an airspace scenario generator and details how to apply a simple stochastic motion model to the aircraft, if the goal is to generate temporal data. While there is no statistical measure of similarity between this and real-world data, the behaviors are derived from observed real-world behaviors such that it details some minimal representation of airspace scenarios. Of particular note within this entire scenario-generation process is an algorithm to procedurally generate randomized polygons with the capability to cultivate certain traits or behaviors in the polygon. While these contributions are not directly linked to ML or statistical applications, they provide a means to generate datasets with features that can be relevant in future research projects.

1.3 – Challenges

This thesis explores a domain—centralized and comprehensive DL navigational systems—where there is little-to-no existing relevant research. There are a number of technical, logistical, and directional challenges that include determining:

1. the structure of a navigational solution;
2. how to collect sufficient navigational feature data to train and test a model;
3. how to discretize, and otherwise pre-process, the collected datasets;
4. whether to use supervised, semi-supervised, or unsupervised learning and then determining how to evaluate the model given the selected approach;
5. what types of neural networks can be used to model navigational features;
6. what types of neural networks can be used to transform modeled navigational features into a navigational solution; and
7. how to analyze the results of the implemented architectural models and compare them to each other.

While the solutions to some of these have already been introduced, the specifics were unknown at the start of this thesis and were eventually selected as the best option when accounting for the scope and the level of detail in the research

1.4 – Chapter Summaries

Material being covered has been detailed through the introduction up till now. This section will then provide the organization of the remaining chapters with respect to the topics, contributions, and solutions to the challenges being addressed and any current related works to considerations to those contents.

Related works are detailed in the following Chapter 2, where a few examples detailing other applications of DL architectures that focus on navigation are mentioned. Additionally, this thesis introduces new functions to evaluate and train models. Therefore, comparable existing methodologies are also discussed here.

Input datasets for the experiments are simulated, and the specifications of these procedural scenarios and how they are pre-processed for ML inputs is detailed in Chapter 3. As part of the specifications for how scenarios are procedurally generated, the specifications for how the navigational solutions are calculated from the navigational features of a space are also given. This chapter also includes some details some rationalizations for the pre-processing decisions and how they are viable for navigation on a WG84 ellipsoid—the Earth's type of ellipsoid—surfaces.

Chapter 4 details an exploration of the architectural approaches detailed in the experiments, describing the selection process, properties of the architectures that make them viable, and providing simple descriptions of the architectural mechanics for readers that may be unfamiliar with DL architectures. This includes granular details such as the structures of the inputs and outputs as well as optimization functions used to train the architectures.

Novel statistical measures and metrics are used to train and evaluate the developed architectures. The technical details of these and their use-cases are presented in Chapter 5. This notably details a new loss function, accuracy metric, and a means by which to evaluate the expected quality of a model's predictions. Additional expansions and use cases of these functions are proposed at the end of the paper in Future Works in Chapter 10.

Details of the experimental environment, experiments and results are explored in Chapter 6, Chapter 7, and Chapter 8, respectively. These provide the specifications of the explored architectures, the training processes, how the model learnt over time, issues encountered while training, and an analysis of the realized predictions from the trained models. These experiments cover modeling navigational features and transforming this information into navigational solutions specifically for spatial data. Temporal-spatial data is briefly explored in Chapter 9, where similar experiments are performed on temporalized scenario data to observe how well the spatial architectures can potentially extend into temporal space. In general, the spatial architectures empirically demonstrate successful results, though there is still room for improvement. The explored temporal architecture is, however, unsuccessful, though it still yields potentially useful information. The complete conclusions are detailed fully in Chapter 9. Finally, Chapter 10 proposes some potential future options for extending this research, ways to improve upon some of the methodologies, or approaches to make the technology more applicable to real-world problems.

Chapter 2 Related Work

Navigation is a popular and broad domain within the realm of ML. Egocentric DL architectures that utilize a mix of reinforcement learning (*deep reinforcement*) and supervised learning are especially prevalent in these domains, and are used in both air and ground-based vehicles and robotics.

Sophisticated autonomous vehicle systems use Convolutional Neural Networks (CNNs) for a nonlinear analysis of relative spatial navigational feature data and Recurrent Neural Networks (RNNs) for processing text and video data, and have access to large datasets used to effectively train these autonomous behaviors to be capable of safe navigation (Grigorescu, et al., 2020). Autonomous pathing is made with multiple steps, detailing a general mission plan, behavior planner, and motion planner is trained with deep reinforcement learning using either *imitation learning* that tries to learn human behavior from recorded experiences and *inverse reinforcement learning* where an adaptive objective function is modeled from human behaviors (ibid).

Simulated datasets have historic use in the domains of training baseline egocentric autonomous agents to navigate a space. In the realm of robotics that detail simpler spaces, the navigable domain is completely understood and so deep reinforcement can be applied to simulated agents that can learn how to behave before being integrated into a real-world autonomous and mobile robot (e.g., Tai et al., 2017). Once the model controlling the autonomous decisions has been sufficiently trained, the software can be installed onto integrated systems that can then use these trained networks in real-world problem spaces.

Deep reinforcement can also be used for unmanned aerial vehicle (UAV) path planning (e.g., Rückin, et al., 2021) (e.g., Theile, et al., 2021). Related to navigation in continuous spaces, coverage path planning (CPP) details an ordering of points in a space to sequentially path towards (Theile, et al., 2021). It is demonstrated that a simulated UAV can use deep reinforcement learning to train a Double Deep Q-Network (DQNN) architecture, used to identify a CPP in a small discrete space for the most effective route per an optimization function (ibid). This demonstration further details two resolutions of egocentric views—full local resolution and a compressed global resolution—of navigational features modeled by CNN layers to produce these determinations (ibid).

The proposed NBTE measures are a novel approach that accounts for both spatial relativity and bias in the information. NBTE Loss is specifically used as the regression loss function when training implementations of the explored DL architectural approaches. However, Cosine Similarity (aka Cosine Proximity or Congruence Coefficient) is a traditional regression loss function with a similar trait: the capability to model loss with respect to the spatial distributions of data. Cosine Similarity is defined as the Euclidean normalization of the dot product between two vectors, detailing an output range of $[-1,1]$. To explain what these values correlate to: -1 indicates identical vectors ($\cos 0°$), 0 indicates orthogonal vectors ($\cos 90°$), and 1 indicates opposite vectors ($\cos 180°$). The standard definition of Cosine Similarity s is defined as:

$$s(\vec{a},\vec{b}) = \frac{\vec{a}\cdot\vec{b}}{\|\vec{a}\|\|\vec{b}\|} = \frac{\sum \vec{a}_i\vec{b}_i}{\sqrt{\sum \vec{a}_i^2}\sqrt{\sum \vec{b}_i^2}} \tag{2-1}$$

Where $\vec{a}$ and $\vec{b}$ are then two n-dimensional vectors being compared. While traditionally used for comparing two n-dimensional vectors, the implementation could be expanded to compare two n-dimensional arrays to be made relevant in comparing two avoidance heatmaps against each other.

This modified definition of the Cosine Similarity to support *n*-dimensional arrays is given in Eq. (2.3-2).

$$s(A, B) = \frac{A \cdot B}{\|A\|\|B\|} = \frac{\sum A_{i_1,i_2,\dots i_n} B_{i_1,i_2,\dots i_n}}{\sqrt{\sum A^2_{i_1,i_2,\dots i_n}} \sqrt{\sum B^2_{i_1,i_2,\dots i_n}}} \qquad (2\text{-}2)$$

Where A and B are then two *n*-dimensional arrays. Alternatively, *n*-dimensional arrays can be flattened to vectors, in which the traditional definition of Eq. (2.3-1) would apply. However, if there is both bias in the values in the space and it is equally important to model all values accurately, then the Cosine Similarity measure becomes infeasible. This is due to a simple strategy to identify a local loss minimum in the Cosine Similarity function: apply a global fill that minimizes the loss from the common values and ignores the loss incurred by the uncommon values. Therefore, the NBTE Loss function is selected over the Cosine Similarity function.

Chapter 3 Airspace Scenario Dataset Generator

The *airspace scenario dataset generator* is the exclusive source of the datasets used for exploring the proposed architectures and methodologies. It details two sequential processes: scenario generation and scenario pre-processing. *Scenario generation* details the process of procedurally producing *airspace scenarios*, which represent some continuous, navigable space that is populated with airspace navigational features and has some corresponding, deterministic navigational solution. *Scenario pre-processing* then takes the information provided in an airspace scenario, and readies it for machine learning by discretizing the space. The term s*cenario* is used to contextually denote either the continuous or discretized space, as both are directly related. An adjacent component of the scenario generator is the *navigational solution calculator*, which deterministically derives a navigational solution from a continuous scenario.

This chapter describes the specifications for procedurally generating these scenarios, detailing the broad objectives of the procedural generation of scenarios, the dimensional measures, discretization process, variation specifications, and how to calculate the navigational solutions. Objectives detail the directional goals of what the data should generally look like or do, where the remainder of the chapter will be detailing how these objectives are satisfied. Dimensional measures inform of the sizes of space that will be used and provide measurement references to put the sizes of the navigational features into context. The discretization process then outlines how the generated scenarios can be restructured into a DL compatible data structure. Variation specifications will describe the ways in which scenario data can differ between two scenarios. Additionally, there will be a brief mention of how the navigational solutions be realistically applied to real-world spaces.

3.1 – Objectives

There are a number of objectives in designing how the scenarios are produced:

1. Scenarios should have identical dimensional measures. This serves to limit the focus of the experiments to the task of modeling navigational features and transforming them into navigational solutions. Developing more complex architectures to handle spaces with variable dimensional measures is not explored here.
2. Scenarios should entail multiple types of navigational features. This problem is with reference to modeling the navigational features of airspaces, which can have many considerations such as airfields, aircraft, no-fly zones, and weather patterns. Thus, multiple features emulate a multifactored consideration of the navigable space.
3. Navigational features should detail variety in their properties. This extends to the distributions of counts, positions, geometries (if applicable), and other attributes of each feature in the space. This is emulates some domain pre-processing, where input information can be assumed to have identified sufficiently similar features. Reducing the type of entities can allow for more efficient training, when the model parameters or the optimizer are dependent on the input sizes, given there are fewer features to update and so it is then easier to find how to best solve the objective function.
4. Variety in the navigational features should have some stochastic elements. This is to add complexity to feature-behaviors to make it more difficult for a model to predict the solution independently of the navigational feature data. E.g., it might be easy for a model to innately know where the airfields are if they are consistently positioned in the same places across

each scenario without actually checking. That is, the model must learn the behaviors of the features and not the behaviors of the space.

5. Navigational features should be generated beyond the range of the space that encapsulates the modeled navigational features—the *captured* spaced—by the model such that they can potentially add some navigational avoidance to the borders of the captured space without the source of the avoidance being captured by the model. Such impacts serve to disincentivize traveling where there are potential unknown features nearby.
6. Navigational features should emulate features realistic of the given characterization of an airspace. While this is partly detailed in the previous points, this point identifies that the generated scenarios should, to some minimal extent, represent an airspace.
7. Navigational solutions should be interpretable of the navigational features. As these experiments are meant to detail some initial research, the material presented should be clearly inferable as to whether or not the trained models detail a successful approach.

The example in Figure 3-1 scenario details some navigable space populated by the full variety of navigational features being produced: airfields, aircraft, type A and B miscellaneous hazards, no-fly zones, and weather fronts. Note that while most of these features are interpretable features that directly emulate some airspace characteristic, the two types of miscellaneous hazards (A and B) are ambiguous points of some additional navigational influence that represent either another distinct type of object or be some aggregated data whose state correlates to some amount of navigational avoidance. That is, the miscellaneous hazards are added to further increase the variations of the scenario. All of the navigational features are varied in that their positions and counts are determined as random realizations of some distributions, and the no-fly zones and weather fronts, which denote some area, are additionally varied with random geometries.

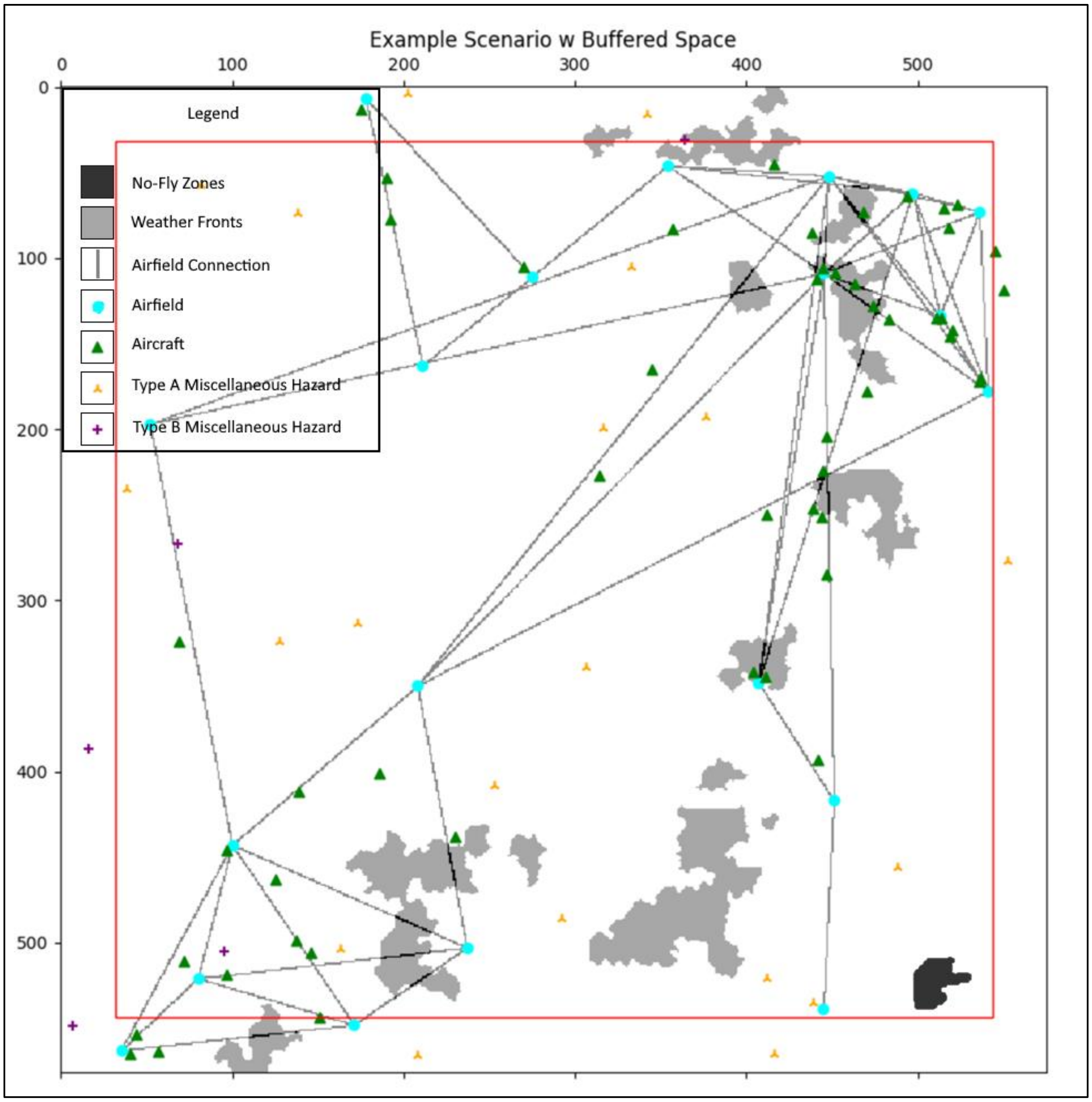

Figure 3-1 – Visualized Scenario Example

Most features are independent of each other. However, airfields and aircraft are the exceptions. The aircraft features are dependent on the airfield features. Airfields determine, relative to their collective arrangement, a number of connections that link, directly or indirectly, every airfield together. Aircraft, when being procedurally produced, are then distributed along these airfield connections. These relations will be observed to see if the resulting autonomous agents may try to learn them to create navigational solutions where the positions of aircraft would be generally predicted to either be along these paths. Note that these connections are not explicitly provided to the model for training; the connections linking airfields must be inferred by the autonomous agent, if they are to be learnt.

While all navigational features simulated are typically constrained to the space of the scenario, the scenario details two regions: an outer ring of uncaptured space and an inner area of captured space, indicated by the outline of the red-square. *Captured* space denotes the regions of space that will be discretized and used as the model input to create the navigational solutions. *Uncaptured* space, in comparison, are then regions of space that will not be discretized. As the navigational features can exist in uncaptured space, they still produce an impact on the navigation solution: their gradient of avoidance. This gradient can extend across the captured-uncaptured boundary, and thus there is a controlled level of unobserved influence on the captured regions. This is useful for training autonomous agents to conceptually learn that just because navigable features near a border of their known space cannot be detected that does not mean that they do not exist. This is another type of relation that will be observed for in the resulting autonomous solutions.

3.2 – Dimensional Specifications

Each generated scenario details a captured area of 512×512 kilometers with a uniform padding of 32 kilometers that defines the total area of the produced scenario as 576×576 kilometers. For the purposes of this research, simulated spaces will be treated as a flat cartesian plane—the consequences of which are explored in §3.4 – Measuring Spatial Error of Navigational Solutions. Therefore, to be captured in the space, navigational features must be positioned in the domain $(x, y) \in [32,544]^2$. For navigational features with an area, only those points that overlap with that domain are captured. For a frame of reference, it is approximately 282 kilometers to travel from Will Rogers World Airport (ICAO: KOKC) in Oklahoma City, Oklahoma to Dallas/Fort Worth International Airport (ICAO: KDFW).

These identified measures are limited to an approximation of the latitudinal and longitudinal axes; altitude is not considered in this step of the research. However, this does not mean altitude could not, in the future, be modeled. The explored architectures are dimensionally scalable, and could be extended to work along another axis. Alternatively, multiple spatial maps could be made identifying the navigational features at different altitudinal steps. Then, the navigational solution for each step of altitude could be computed and then the gradients of avoidances interpolated between the steps to produce the 3-dimensional navigational space. Because these ways are largely extensions of the approaches being explored, it was thus rationalized as being acceptable to reduce the altitudinal axis from consideration for this research.

3.3 – Discretizing Process

The discretization process can be described as projecting a grid of the desired resolution onto the navigable surface and then mapping the navigable features into the nearest cell within that grid. For this research, a 1 km $\times$ 1 km resolution is used, producing a 512×512 grid given the 512 km $\times$ 512 km measures of the captured space. This grid is projected onto the navigable area and the positions navigable features are mapped to a 1-dimensional boolean vector that details the presence of navigable features within that cell of the projected grid. This produces a 3-dimensional array of information that will be the input for the experimented models. Elements within these arrays are then constrained to the binary domain $\{0,1\}$.

Attributes of the navigable features could also be included to expand the features of the input data, producing a 4-dimensional array. While for small sizes this might be acceptable, or even beneficial, it comes with a cost: dimensionality scales the computational complexities by an exponential factor. Thus, for large base sizes such as 512×512, extending the dimensionality is very computationally expensive. For that reason, the simple 3-dimensional array as a minimally viable approach that could potentially be improved by adding further detail if needed.

Alternatively, the vector of indicators could be converted to a vector of population size for each feature, where instead of a presence-indicator for each feature type it details the quantity of each type. However, in the characterized domain of airspaces, navigational features are positionally distributed such that overlap between any two randomly sampled scenarios is improbable. Therefore, this again details a minimally viable approach that could potentially be improved by expansion. However, it is controlled for in this problem space as it was ultimately concluded that, due to the rarity of occurrence, it would not significantly impact results.

There are 6 types of navigable features being tracked, as indicated by in Figure 1. Therefore, the result is a 3-dimensional array with the measures $512 \times 512 \times 6$) in height, width, and then channel (HWC) format. The channel ordering of features is as follows: 1) Airfields, 2) Aircraft, 3) Miscellaneous Type A Hazards, 4) Miscellaneous Type B Hazards, 5) No-Fly Zones, and 6) Weather Fronts.

For example, given a boolean vector with values $[0,1,0,0,0,1]$ it is indicated that there is at least one aircraft and a weather front present for the position in the grid.

3.4 – Measuring Spatial Error of Navigational Solutions

For many smaller scale domains, a 1:1 mapping from a cartesian plane to the real-world space is effectively an accurate representation. However, as the size of the space is increased, the curvature of the Earth becomes increasingly important to account for. Because of the large measures of the space being considered for the scenarios and the assertion that the simulated space can be treated as a cartesian space, the error between the simulated and real is briefly explored. While there is some error between the area of the navigational solution for the simulated measures and the surface area of the Earth that the navigational solution would be projected, the difference is still marginal enough to treat airspaces as cartesian planes for this research.

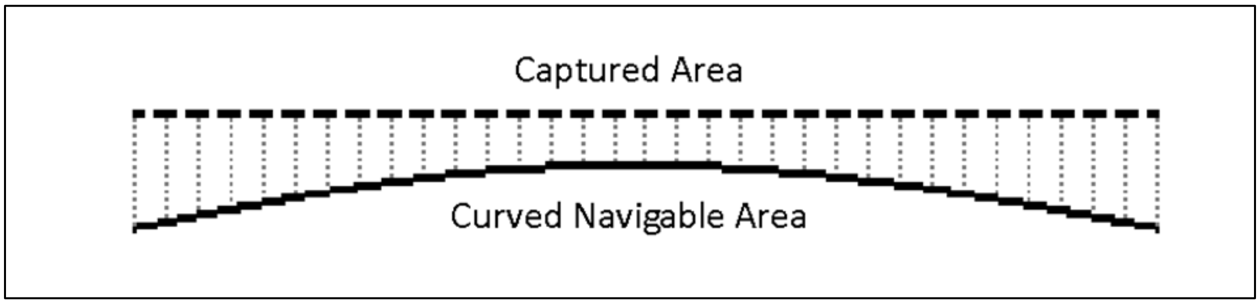

Figure 3-2 – Visualization of Projection Error

To demonstrate this, consider the following example that details how the proposed 512×512 navigational solution is proportional to the underlying curved surface upon which the navigational solution is orthogonally projected onto. This is an approximate calculation, given the

Earth is a WGS84 ellipsoid; it is not perfectly spherical. Between $0°$ and $90°$ of latitude, the conversion rate of kilometers per degree of latitude is increased from 110.574 kilometers per degree to 111.694 kilometers per degree: an increase of approximately $+1.0129\%$. Note that this denotes the latitudinal change from an equator to a pole, which is an exact surface-distance of $10{,}000$ kilometers; this span is significantly more than the example 512 kilometers. Therefore, only the initial conversion rate will be used, for latitude centered about the equator, to simplify the calculation of error as the resulting change of conversion rate will be negligible.

To calculate the error in proportion, first compute the arc angles of the navigable area with respect to the Earth. Then, compute the arc lengths of the previous arc angles. These arc lengths can be extrapolated to provide the approximate surface-area of the navigable area, which can then be compared with the area of the navigable solution. The terms ϕ, λ denote the measure of latitude/longitude (in degrees), respectively. Similarly, d_ϕ, d_λ will be used to denote the measures of the navigable solution (in kilometers) with respect to the latitude/longitude axes. Then, Φ/Λ will denote the conversion of kilometers per degrees (km/deg) at a latitude of 0°. Again, these conversion rates are considered constant for this example given the marginal change the actual would provide to the final determinations.

Given are both the distances as $d_\phi = 512\text{ km}, d_\lambda = 512\text{ km}$, derived from the specifications of the captured space produced by the airspace scenario generator, and the conversion rates as $\Phi_{0°} = 110.574\text{ km/deg}, \Lambda_{0°} = 111.320\cos(\lambda)\ \text{km/deg}$. Then the angles ϕ and λ are calculated with Eq. $(3.2\text{-}1)$ and $(3.2\text{-}2)$.

$$\phi = \frac{d_\phi}{\Phi_{0^\circ}} = \frac{512\ \text{km}}{110.574\ \text{km/deg}} \cong 4.63038^\circ \tag{3.2-1}$$

$$\lambda = \frac{d_\lambda}{\Lambda_{0^\circ}} = \frac{512\ \text{km}}{111.320\cos(4.6304^\circ)\ \text{km/deg}} \cong 4.61441^\circ \tag{3.2-2}$$

Where the angle of longitude is 99.66% the angle of latitude, owing to aforementioned unevenness in the curvature of the Earth. The arc length of the surface s denotes the distances (in kilometers) between the latitudinal and longitudinal angles and can be calculated with the semi-major axis radius—which is at 0° of latitude—of the Earth $R_E = 6378.1370\ \text{km}$ with Eq. $(3.2\text{-}3)$ and $(3.2\text{-}4)$.

$$s_\phi = \frac{\pi R_E \phi}{180^\circ} = \frac{\pi(6378.1370\ \text{km})(4.63038^\circ)}{180^\circ} \cong 515.45191\ \text{km} \tag{3.2-3}$$

$$s_\lambda = \frac{\pi R_E \lambda}{180^\circ} = \frac{\pi(6378.1370\ \text{km})(4.6144^\circ)}{180^\circ} \cong 513.67418\ \text{km} \tag{3.2-4}$$

Then the proportion of the projected navigational solution area and the actual navigable area, extrapolating the arc lengths into an arc-surface area, is given in Eq. $(3.2\text{-}5)$.

$$\frac{d_\phi d_\lambda}{s_\phi s_\lambda} = \frac{(512\ \text{km})^2}{(515.45191\ \text{km})(513.67418\ \text{km})} \cong 99.00657\% \tag{3.2-5}$$

Note this proportional error is the aggregated error across the entire space. For an example resolution of the navigational solution at 1 km $\times$ 1 km, which would give a navigational solution with sizes 512×512 for the 512 km $\times$ 512 km area of the captured spatial data, each $1\ \text{km}^2$ cell corresponds to an average $\sim 1.01003\ \text{km}^2$ cell on the navigable surface. This approximation of cell-based error is not exact for each, again given the unevenness of the Earth's curvature, but it details the average skewness.

In conclusion of these evaluations, the calculated level of error was considered acceptable for experimental purposes on identifying navigable solutions; it is acceptable to use a cartesian-plane interpretation of an airspace of the given measures.

3.5 – Variation Specifications

The set of known attributes of navigational features is exclusively their respective position(s). If they are sufficiently large such that they are likely to expand across multiple points in the discretized space, they are considered as polygonal-based features and their attributes denote the positions in which they extend at least partially contained in. Point-based features are the opposite; the dimensional measures of these features are small enough to reasonably determine they will typically not occupy more than a single cell per the resolution of the discretized space. Of the included navigational features in this problem space, the point-based features include: airfields, aircraft, and the miscellaneous entity types. Then, the polygonal-based features include: no-fly zones and weather fronts. These navigable features are then be varied by changing their positions and the count of their instances for each scenario. For the polygonal-based features, note that the variable positions detail variable shapes and sizes of the polygon itself, in additional to an offset of the polygon's bounding box with respect to the space of the scenario.

3.5.1 – Positional and Count Specifications

Given the notation $x_{\text{pos}} \sim U(a,b)$ to mean the realized random variable detailing the position of the navigational feature x is in the continuous uniform distribution from the interval $[a,b]$, then let the notation $\mathbb{U}(a,b)^n$ detail a coordinate position in n dimensional space where the offset along each axis of the coordinate is a random variable in the distribution $U(a,b)$. Then the counts, indicated by taking the magnitude of a set of the features, and positions, with respect to the origin of the scenario, of the sets of navigational features for each scenario are:

$$|\text{Airfields}| \sim N(\mu = 16, \sigma = 4) \tag{3.5.1-1}$$

$$\forall x \in \text{Airfields}\left(x_{\text{pos}} \sim \mathbb{U}(0\text{ km}, 575\text{ km})^2\right) \tag{3.5.1-2}$$

$$|\text{Aircraft}| \sim N(\mu = 48, \sigma = 8) \tag{3.5.1-3}$$

Where the positions of aircraft are dependent on airfields, which is detailed later in this section.

$$|\text{Misc. Entities Type A}| = 20 \tag{3.5.1-4}$$

$$\forall x \in \text{Misc. Entities Type A}\left(x_{\text{pos}} \sim \mathbb{U}(0\text{ km}, 575\text{ km})^2\right) \tag{3.5.1-5}$$

$$|\text{Misc. Entities Type B}| = 5 \tag{3.5.1-6}$$

$$\forall x \in \text{Misc. Entities Type B}\left(x_{\text{pos}} \sim \mathbb{U}(0\text{ km}, 575\text{ km})^2\right) \tag{3.5.1-7}$$

No-Fly Zones and Weather Fronts are also positioned differently, as they detail a geometry that is offset with respect to the origin of that bounding box (top-left corner) for their geometry. Furthermore, this origin can actually be positioned such that part of the polygon exists beyond even the unobserved region of a scenario, as it is possible the bounding box can exceed the measures of the scenario. The notation x_d is used to denote the measure of the bounding box along dimension d of a polygon-based entity (measured in kilometers). Additionally, the notation $|x| \sim P(X) = Y$ is used to denote that some event x is to occur with a count of Y realizations using the probability distribution $P(X)$.

$$|No - Fly\ Zone| \sim P(0.3) = 1 \tag{3.5.1-8}$$

$$\text{No-Fly Zone}_{\text{pos}} \sim \mathbb{U}\begin{pmatrix} 0\text{ km} - 0.55\text{No-Fly Zone}_{\text{d}}, \\ 576\text{ km} - 0.45\text{No-Fly Zone}_{\text{d}} \end{pmatrix}^2 \tag{3.5.1-9}$$

$$|Weather\ Front| \sim P(0.3) = 1 \tag{3.5.1-10}$$

$$\text{Weather Front}_{\text{pos}} \sim \mathbb{U}\begin{pmatrix} 0\text{ km} - 0.55\text{Weather Front}_{\text{d}}, \\ 576\text{ km} - 0.45\text{Weather Front}_{\text{d}} \end{pmatrix}^2 \tag{3.5.1-11}$$

Which additionally allows for incomplete information of these polygonal-based features to be simulated. Again, note the displacement of the polygon-based features is with respect to the origin of their bounding box, not the center of the box. Thus, the positional offset is non-symmetrical.

Aircraft are unique in that they are positionally dependent on the airfields. The placement of airfields deterministically produces a series of connections between the airfields. Aircraft are positioned by selecting a random connection and are then distributed along this connection. The specifics of how they are distributed were deemed acceptable after observing the simulated behaviors and deciding these behaviors were sufficiently similar to observations made of real aircraft. This distribution of the aircraft most importantly details some random deviation from the direct connection line that adds controlled noise to the simulated aircraft features, which further details nonlinear behaviors and thus adds more complexity to the scenario.

When being placed along a connection, three random variables are individually used to determine the relative progress from the originating airfield to the destination airfield (x), the magnitude at which the position deviates from the direct line (y), and the direction to apply the deviation (v_r). The distribution details greater potential for displacement as the aircraft approaches the midpoint of the flightpath connection, with a maximum potential magnitude of deviation of 25 km. Then the noise function g is then given as:

$$g(x, y, \vec{v}_r) = \|\vec{v}_r\| 25 \text{ km } y \max(\sin(\pi x), 0.2) \tag{3.5.1-12}$$

$$x \sim U(0,1) \tag{3.5.1-13}$$

$$y \sim U(0,1) \tag{3.5.1-14}$$

$$\vec{v}_r \sim \mathbb{U}(-1,1)^2 \tag{3.5.1-15}$$

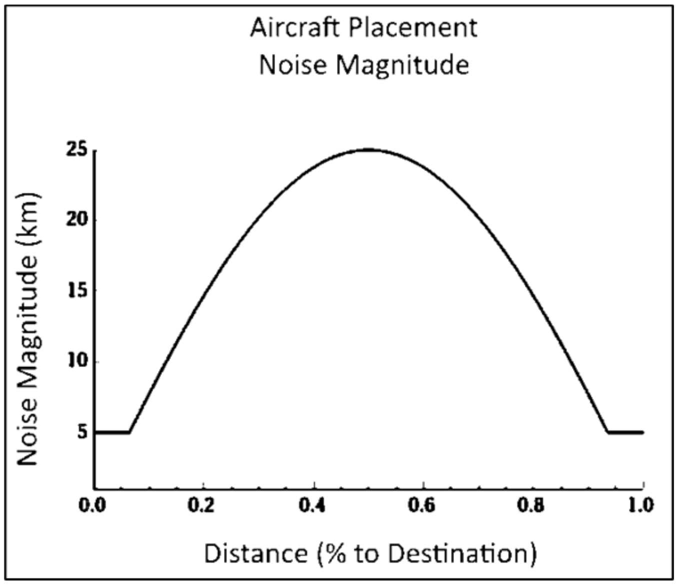

Figure 3-3 – Distribution of Aircraft Position Maximum Noise

To calculate the initial placement, this noise is added to the calculated position of the decided point directly on the airfield connection—determined by the relative progress random variable x—between the positions of both the originating airfield ($\vec{a}$) and the destination airfield ($\vec{b}$). Then the function determining the position of an individual aircraft f is:

$$f(\vec{a}, \vec{b}, x, y, \vec{v}_r) = g(x, y, \vec{v}_r), +x\left\|\vec{b} - \vec{a}\right\| + \vec{a} \tag{3.5.1-16}$$

A visualization of the potential area an aircraft could be placed in between two airfields 100 km apart is given in Figure 3-4.

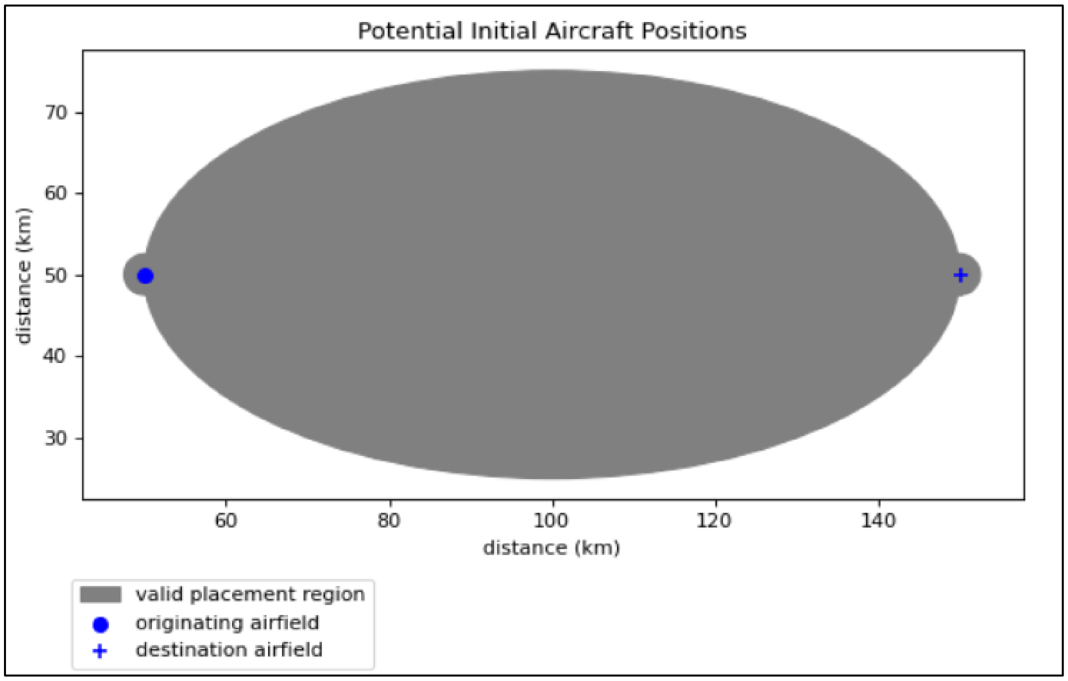

Figure 3-4 – Potential Aircraft Position Visualization

While the maximum potential of this noise function details a large area, the expected realization of deviation is greatly reduced from the maximum. This is due to the mean of the random variable y giving a mean magnitude of 12.5 km, down from the maximum potential of 25 km.

Additionally, the random directional vector u_r further reduces the expected deviation from the direct connection path itself, not the progress point of the aircraft along the line, as it is improbable that u_r will be perfectly orthogonal to the direct connection line.

3.5.2 – Geometric Variation Specifications

To further improve upon the variations between scenarios, polygonal-based features have randomized geometries. This producing various edges, sizes, and patterns, where certain traits in the generation process can be encouraged to produce consistent emergent behaviors.

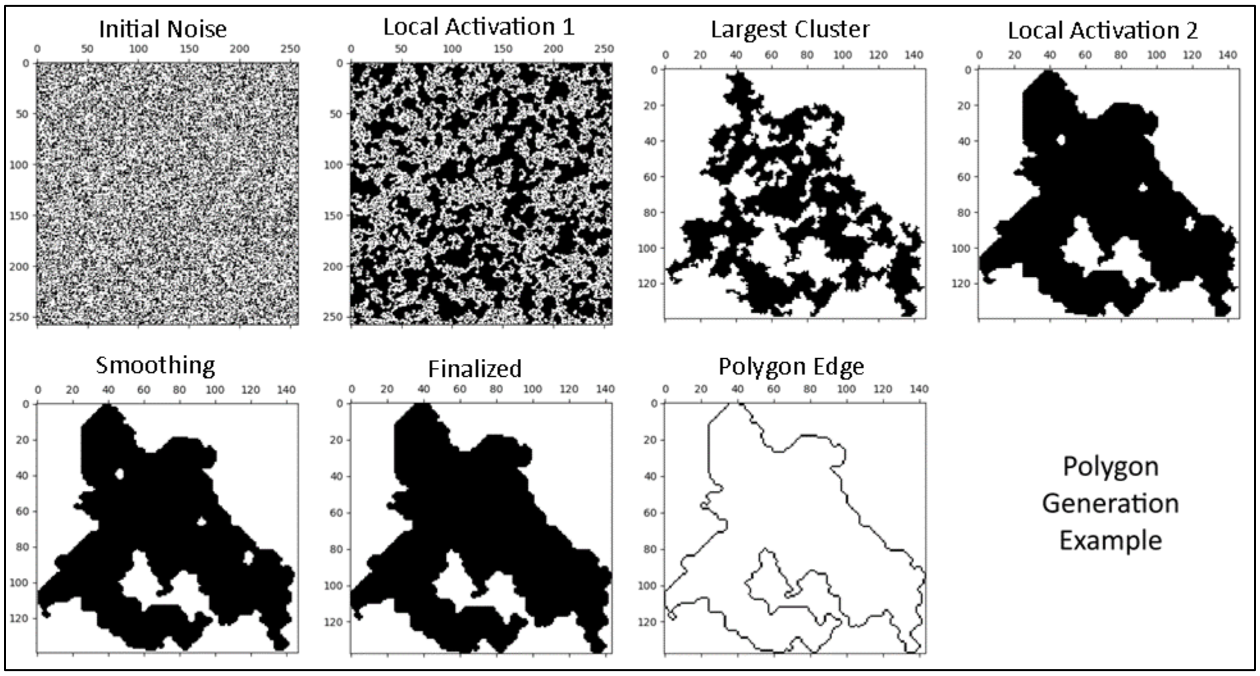

Figure 3-5 – Polygon Generation Example (Top-Left to Bottom-Right)

This process details 5 steps to generate a polygon: 1) Initial Noise Activation, 2) 1st Local Activation, 3) Isolate the Largest Contiguous Cluster, 4) 2nd Local Activation and 5) Smoothing

Step 1 – Initial Noise Activation

This phase creates a binary matrix m where each element is randomly made to be active or inert—1 or 0. The initial determinations are made with respect to a sensitivity variable that details the probabilistic likelihood of each element being activated.

$$\forall(i,j) \in m\left(m_{i,j} = U(0,1) \geq \text{sensitivity}\right) \tag{3.5.2-1}$$

$$0 < \text{sensitivity} < 1 \tag{3.5.2-2}$$

Step 2 – Local Activation 1

Each element in the initial activated noise matrix is then processed in the conditional chain-update operation to build out clusters of activated pixels, thus deriving a matrix with more activated elements, which ideally begin to form chains of connected clusters throughout the matrix. This activation step is made aggressive with respect to the sensitivity; the more initially activated elements, the higher the chances of an aggressive local activation are for causing a chain reaction that activates most of, if not the entire, matrix. Therefore, this step typically produces a series of clusters that, within each cluster, detail multiple small bodies that are typically only attached by a few elements to each other.

Step 3 – Isolate the Largest Contiguous Cluster

All the contiguous clusters (that detail of a chain of elements that are adjacent to each other) are explored and the largest cluster, by the number of elements it contains, is selected and all other clusters are discarded. This produces a matrix where the elements of the selected cluster are the only activated elements in the matrix. This will allow for more aggressive localized activations, as it reduces the number of activated elements in the matrix that could result in a critical chain reaction that activate all elements in the matrix.

Step 4 – Local Activation 2

While the first local activation helped create some sprawling connected clusters in the initial matrix of noise, the second local activation will produce most of the final edges of the polygon using the current contiguous cluster as the base.

Step 5 – Smoothing

This step is used to remove long, thin stretches of activated pixels that could not grow into clusters. Note that this step has a chance of splitting clusters, but the greater the sensitivity and the more aggressive the thresholds used for the local activation steps, the less likely this becomes. Clusters that are split apart will be still treated as the same polygon, however.

3.5.2.1 – Conditional Chain-Update Algorithm

The algorithm used to expand activated clusters and smooth the polygon is a custom algorithm labeled as the Conditional Chain-Update Algorithm. This focuses on each element in the matrix and determines, with respect to the local neighbors of that element, if the element should be updated to the target value. If the element is updated, a chain-reaction occurs that applies the process to neighbors that have the potential to also be updated. The count of local neighbors of the targeted value required to propagate an update is a random variable that is drawn with uniform probability from a list of potential threshold values. The selected threshold is additionally used for all the chain-updates actuated. This provides for weighted variations in the resulting shape of the polygon that can encourage certain traits. Note that neighbors, for this algorithm, include corners. Additionally, the order of both explored root elements and neighbors occur in a Top-Left to Bottom-Right (TLBR) approach. That is, the TLBR approach iterates across each column in a row, left-to-right, before proceeding to the next row. Finally, note that cells along an edge simply have fewer neighbors than other cells. The detailed algorithm is provided in in Algorithm 3-1.

Algorithm 3-1: Conditional-Chain Update

Input: binary 2d array *matrix*, binary scalar *target-value*, and an integer list of *thresholds*.
Output: an updated binary 2d array matrix.

```
(1) for each cell root ∈ matrix (TLBR Iteration):
(2)     if $root_{value}$ = target-value then:
(3)         continue to the next root
(4)     initialize integer scalar threshold ← random element ∈ thresholds
(5)     initialize cell list queued-updates ← [root]
(6)     for each cell node ∈ queued-updates :
(7)         initialize cell list candidates ← [   ]
(8)         initialize integer scalar counter ← 0
(9)         for each neighbor ∈ $neighbors_{node}$ (TLBR Iteration):
(10)            if $neighbor_{value}$ = target-value then:
(11)                counter ← counter +1
(12)                if counter ≥ threshold then:
(13)                    $root_{value}$ ← target-value
(14)                    insert candidates into queued-updates
(15)                    break and continue to the next cell
(16)            else:
(17)                insert neighbor into candidates
(18) end for each
(19) return matrix
```

Algorithm 3-1 - Conditional Chain-Update Algorithm

3.5.2.2 – Conditional Chain-Update Algorithm Example

This section details an example usage of the Conditional-Chain Update Algorithm. In this example, a simple 5×5 matrix is generated, detailed by Figure 3-6, where there are a number of disjoint clusters therein.

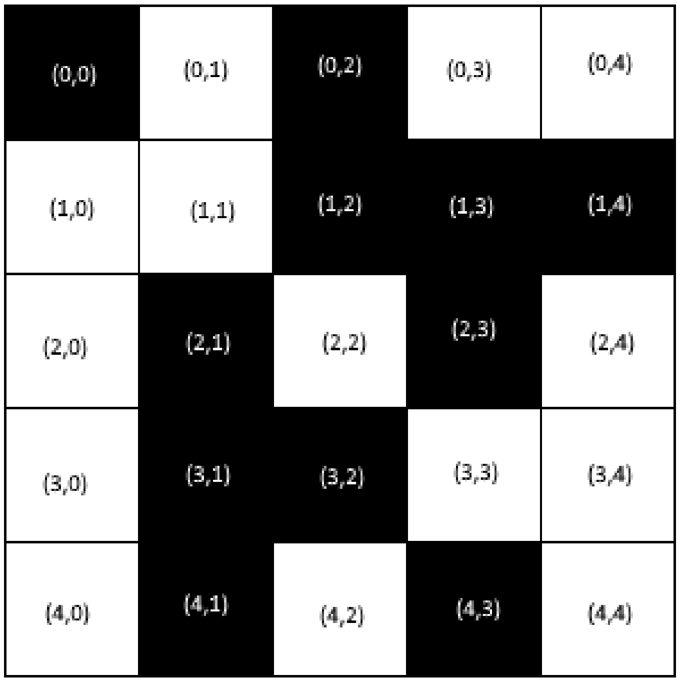

Figure 3-6 – Example Matrix (Black=1, White=0) w/ Coordinate-Labels

These given states detail an example initial state after generating the initial matrix of noise, where the next step is a local activation to connect clusters together to form larger clusters. The example will apply the algorithm with the focus on root $(2,2)$, which is given by Figure 3-6 as being initially inert. As with the Local Activation 1 phase, the target values will be the activated value indicator: 1. The process is detailed in Figure 3-7 with the immediate impact visualized in Figure 3-8.

Data: *matrix* detailed in Figure 7, a *target-value* of 1, and the *thresholds* [5,6,7,8]. Additionally, this example will jump to the cell (2,2) for the first example *root*.

Line 1: *root* initialized to (2,2) with a value of 0

Line 2: *root* is inequal to the *target-value* → skip to Line 4

Line 4: *threshold* randomly selected as 6

Line 5: *queued-updates* initialized to [(2,2)]

Line 6: *node* initialized to (2,2)

Line 7: *candidates* initialized to [ ]

Line 8: *counter* initialized to 0

Line 9.a: *neighbors$_{node}$* defined as [(1,1), (1,2), (1,3), (2,1), (2,3), (3,1), (3,2), (3,3)]

Line 9.b: *neighbor* initialized as (1,1) with a value of 0

Line 10: value of *neighbor* is inequal to *target-value* → skip to Line 17

Line 17: *candidates* updated to [(1,1)]; continue to next *neighbor*

The neighbors are continually explored through cell (3,2), at which point the threshold requirement for the update is reached. Note that at this point the state of the *candidates* is still unchanged; no new candidate cells have been found.

Then, continuing:

Line 13: value of *root* is updated to the *target-value*

Line 14: *queued-updates* updated to [(1,1)]; continue to next *node*

And so on until the algorithm concludes.

Figure 3-7 – Conditional Chain-Update Algorithm Example

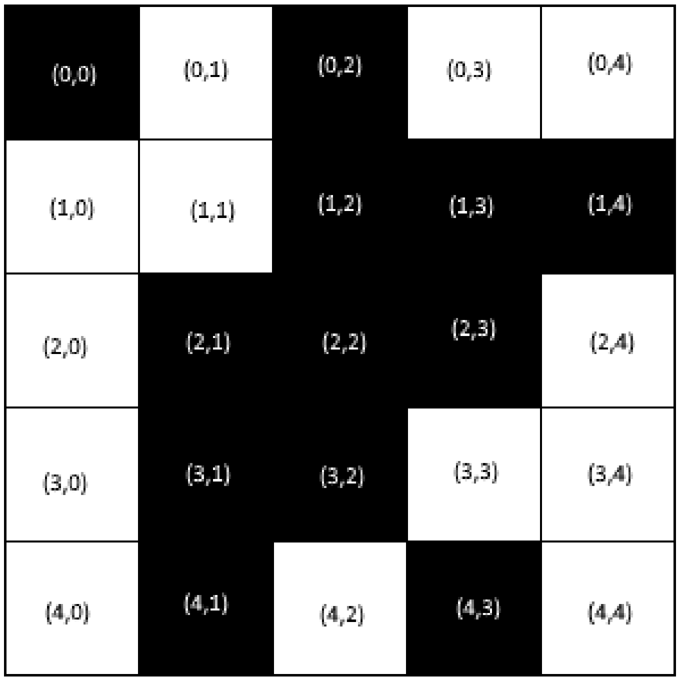

Figure 3-8 – Example of Locally Activated Matrix After Updating (2,2) w/ Coordinate-Labels

3.5.2.2 – Cultivating Geometric Traits

Creating configurations requires balancing the initial noise sensitivity and the configurations of the local activation thresholds. If the sensitivity is too low, then the final polygon will likely be very small. If the sensitivity is too high, the local activations can more easily cause chain reactions that activate most of the matrix. If the first local activation is too passive, the chance of forming chains of clusters is reduced and the final polygon will likely be small or prone to being elongated in some direction. If the first local activation is too aggressive, then many chains will be formed and the likelihood of developing traits like hooks or tails in the polygon are diminished. If the second activation is too passive, the polygon will likely have a very noisy perimeter. If the second activation is too aggressive, the edges of the polygon could be mostly turned into straight lines.

Thresholds for activation steps are constrained to be in the discrete inclusive interval [4,9], where 9 is a special value that indicates an impossible condition as there can be a maximum of 8 neighbors. 4 is also a special value, as it produces straight lines. An example of this is given in Figure 9, where the previous polygon generation is modified such that the second local activation threshold is always 4.

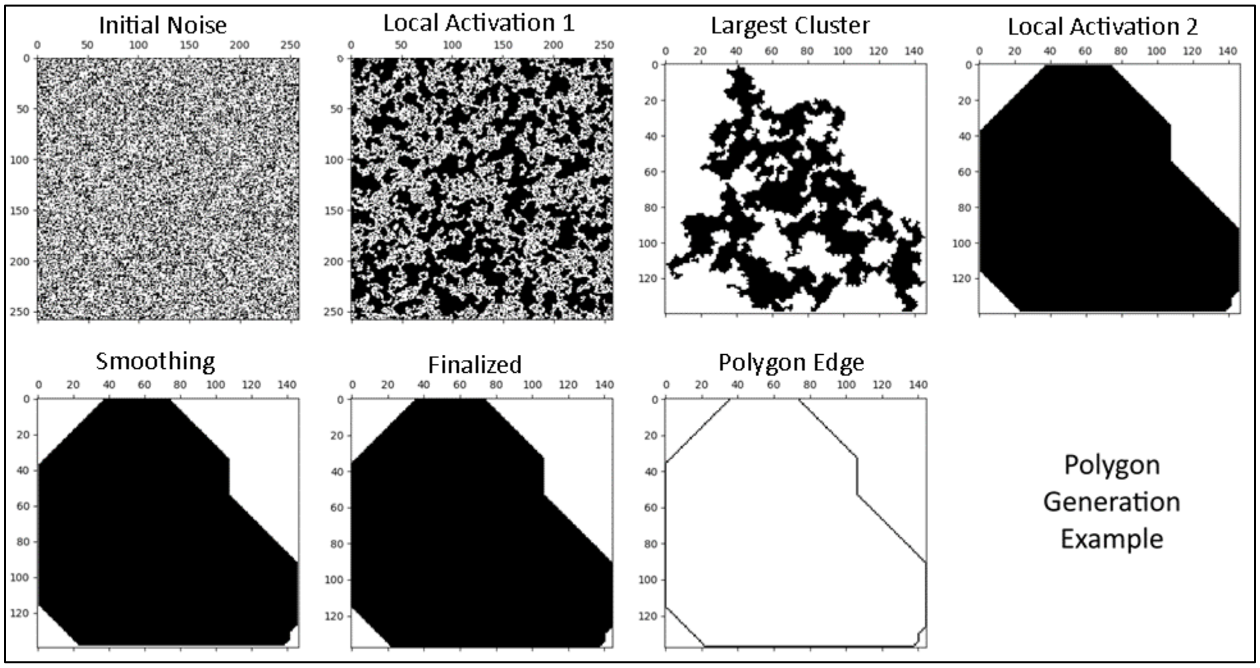

Figure 3-9 – Uniform Local Activation 2 Threshold = 4 Example

Thus, by creating a distribution of thresholds, it can weight the probabilistic behaviors of the formed edges of the polygon to vary between extremely straight and extremely noisy.

Note that the area of the polygon is massively increased between Figure 9 and Figure 4 with the modified thresholds. The lower the thresholds, the greater the area of the polygon. As a result, thresholds for the first activation stage are typically further constrained to be greater-than 4 as otherwise the entire matrix is prone to being activated. For the second activation step, the minimum constraint of 4 is used because a threshold < 4 will result in a perpetual update that consumes all continuous regions of inactivated elements, which will likely result in the entire matrix becoming activated.

3.5.2.3 – No-Fly Zone Configuration

Initial Matrix Size: 200×200
Sensitivity: 40%
Local Activation 1 Thresholds: $[6]$
Local Activation 2 Thresholds: $[4,5,5]$
Smoothing Thresholds: $[5]$

Figure 3-10 – No-Fly Zone Generation Parameters

This configuration reliably produces no-fly zones whose area is approximately within the measures 10×10 and 50×50. Though it theoretically could be upwards of the measures 200×200, given enough samples. The list of thresholds for the first local activation is set to a single value, so that it will always be selected. That value of 6 provides a fairly non-aggressive activation that produces typically smaller shapes. The list of thresholds for the second activation is then designed to select between two slightly more aggressive options: 4 and 5. Note that the threshold 5 is listed twice, giving it twice the weight for being selected than the threshold 4. This encourages for more organic edges, but still allows for the artificially decided boundaries that involve straight lines. These parameters encourage the growth of polygons with small tails and hooks, support some flat edges, and a somewhat bulky body. A series of example No-Fly Zones were generated and provided in Figure 3-11 – Example No-Fly Zones.

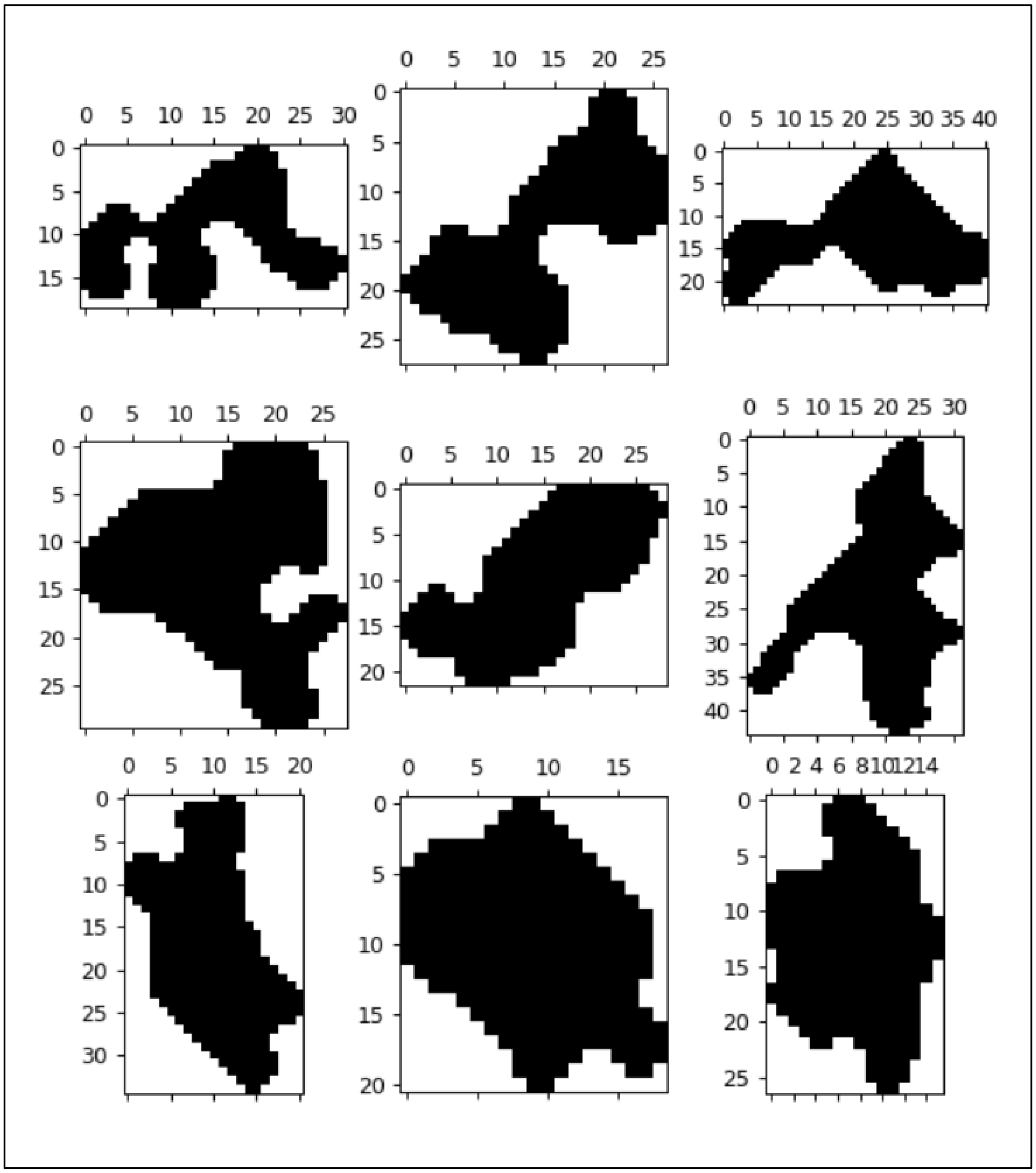

Figure 3-11 – Example No-Fly Zones

3.5.2.4 – Weather Fronts Generation

Weather Fronts differ from No-Fly Zones in that they are composed of a number of individual polygons that makeup some hotspots of inclement activity within a weather front. There are a few extra modifications to the original algorithm for creating a weather front: a directional vector, target length, and cell-spacing parameters are required. Additionally, weather front polygons do not fill gaps that form in the clusters after the activation phases.

Weather fronts are generated by creating a chain of weather cells, indicating regions of inclement conditions within the front, that typically advance in the specified direction. These cells are additionally given a constant spacing such that the generated weather front is not wholly a contiguous cluster of weather cells. The collective bounding box encapsulating all of these cells is used to evaluate the length of the weather front, where the hypotenuse of the bounding box is restricted not to exceed the target length.

The placement of cells is determined by a constant unit vector for the direction the weather front should extend in (u_0), a unit vector of momentum detailing the direction the last cell's placement (u_t), and a random directional vector (v_r) that adds random variance to the advancing direction and shifts momentum. The direction of momentum u_t is with respect to the number of previously placed weather cells t; the direction of momentum is the previous direction of advancement. For the first weather cell being placed at $t = 0$, note that $u_t = u_0$.

These vectors are given relative weights that allow for variation, but ultimately advance the weather cells in the general direction of u_0. These weights are denoted by w_0, w_m, w_r, respectively.

$$w_0 = 0.2, w_m = 0.5, w_r = 0.3 \tag{3.5.2.4-1}$$

$$w_0 + w_t + w_r = 1 \tag{3.5.2.4-2}$$

Note that the sum of weights is 1, so the aggregated sum of directional vectors multiplied by their respective weights will yield a unit vector. Then the direction of advancement for the next cell is given as:

$$u_{t+1} = u_0 w_0 + u_t w_m + |v_r| w_r \tag{3.5.2.4-3}$$

$$v_r \sim \mathbb{U}(0,1)^2$$

The bounding boxes of cells are constant, though the size of the polygon therein may vary. These boxes are square and their side length is given by the variable l_c. Given a target length for the weather front l_w, a matrix m is created with dimensions $n \times n$, which are calculated as:

$$n = \sqrt{2l_w^2} \tag{3.5.2.4-4}$$

Which will provide plenty of room to grow the weather front.

Given the unit vector for the direction of the weather front u_0, the index of origin for the first cell's bounding box p_0 is given as follows:

$$p_0 = \left\lfloor \frac{l_c - u_0 l_w}{2} \right\rfloor \tag{3.5.2.4-5}$$

Where the bounding box is inserted with the position as the top-left corner. Note that the notation of taking the floor of a vector means to truncate any floating-point coordinates into an integer for indexing in the matrix m.

Then, for the remaining cells, given some previous position p_t (the point of insertion for the previous bounding box) and inter-cell spacing constant s, the position of the next weather cell's bounding box p_{t+1} (again, for the top-left corner) is given by:

$$p_{t+1} = p_t + \lfloor s u_{t+1} \rfloor \tag{3.5.2.4-6}$$

Weather cells are repeatedly added in such a fashion to the weather front until either the bounding box of a cell would exceed the size of the matrix or the side-length of the cell, if added to the hypotenuse of the state of the minimal bounding box of current weather front, would exceed the target length.

Let h_t be the length of the hypotenuse of the minimal bounding box of the current weather front. Then, cells are added to the weather front until the following condition is satisfied:

$$l_w \leq l_c + h_t \vee \big(\exists d \in p_{t+1} (d < 1 \vee n < p_{t+1} + l_c)\big) \tag{3.5.2.4-7}$$

Where $d \in p_{t+1}$ is to say for some coordinate position in the point p_{t+1}

Once the weather front is finalized, the matrix can be reduced such that its shape is minimized but it does not exclude any of the added weather cells. This yields the structure of the weather fronts incorporated into the scenario. It is important to remember that these weather cells are treated as a collective polygon, and so their edges are also considered collectively when calculating avoidance; there is no overlap of avoidance values between two weather cells.

3.5.2.5 – Weather Front and Cell Configuration

Weather Front Direction: Bearing $270°$ (North→South)

Weather Cell Spacing: 50

Maximum Weather Front Size: $\sim U(800,1000)$

Maximum Weather Cell Size: 150×150

Sensitivity: $\sim U(0.275, 0.375)$

Local Activation 1 Thresholds: [5,6,7,8]

Local Activation 2 Thresholds: [4,5,5,5,5]

Smoothing Thresholds: [9]

Figure 3-12 – Weather Front Generation Parameters

In contrast to the No-Fly Zones configurations, some settings here are random variables that are realized whenever it is time to produce a new weather front. These distributions allow for greater variations in the resulting generated polygons. Other significant differences here include the thresholds. The first local activation thresholds are, probabilistically, less aggressive than No-Fly Zones, but have the potential for certain regions of the clusters to be expanded more aggressively with a lower minimum threshold. The second local activation thresholds are, however, similar to the No-Fly Zones, except the chances of flatter edges are further reduced. The smoothing thresholds detail disabled smoothing, as it will be impossible to find 9 inert neighbors. This will produce clusters that can be thin or bulky, small or large, flat or noisy, and hooked/forking or straight. This is a large variety of potential features that simulates the large variety of shapes that these weather cell hotspots can represent.

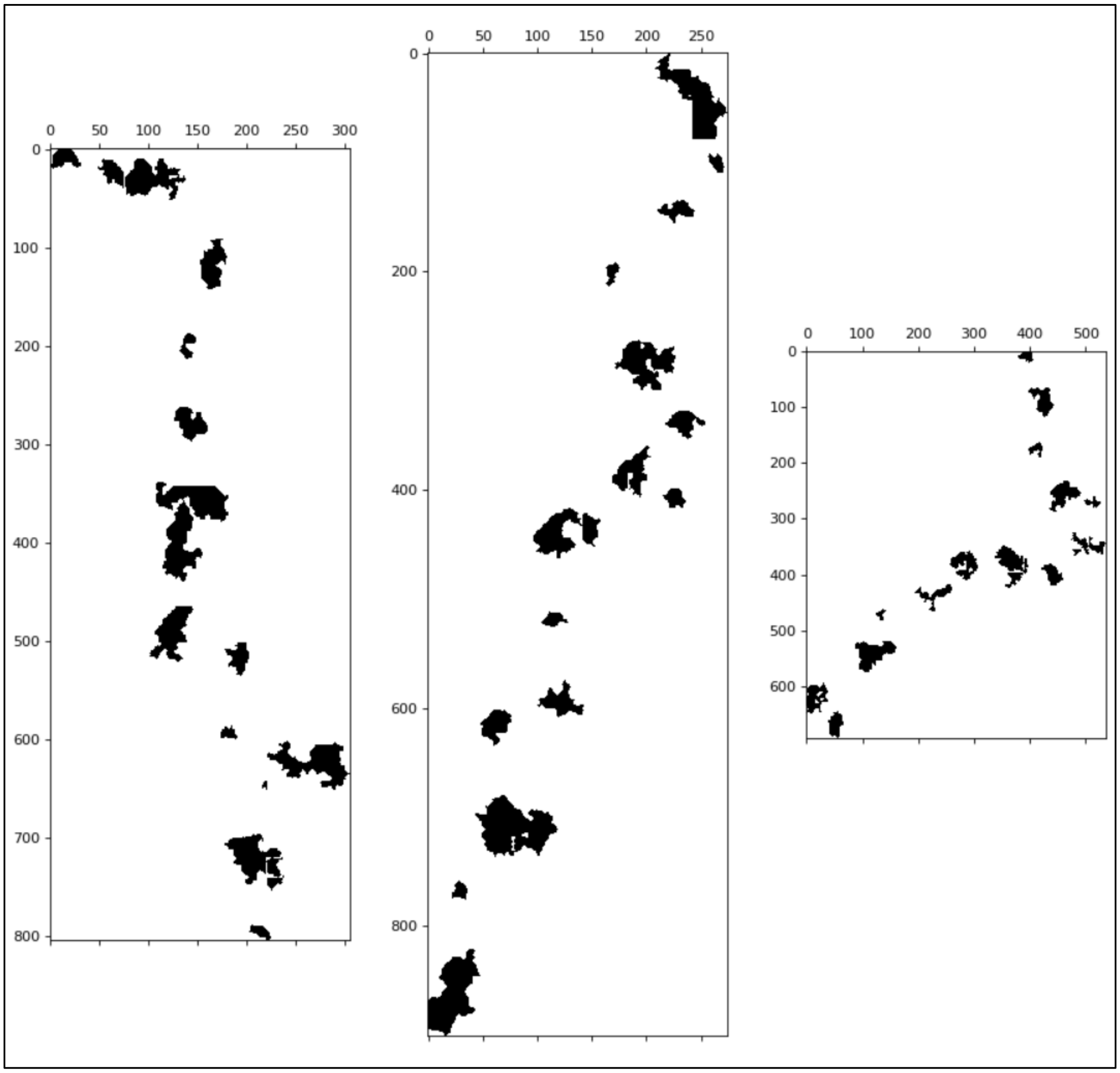

Figure 3-13 – Weather Front Examples

3.5.3 – Airfield Connection Algorithm

Airfields are uniformly distributed across the space, and are subsequently iteratively connected such that all nearby airfields are initially connected and then major hubs begin to form, connecting across longer distances. Using the algorithm for the generated airfields, there are an average 1.55 bi-directional connections per airfield. This averages at 24.8 bi-directional connections per scenario after accounting for the mean airfield count size. Looking at the mean aircraft count size, this comes at to about a 1:1 ratio between the count of aircraft and the count of directional connections, though it is likely multiple aircraft will exist along the same connections given the randomness involved in the selection processes. This end-result of this algorithm is visualized in the earlier Figure 3-1, where the lines established between different airfields represent the airfield connections. This details a series of local connections with the more connected airfields bridging long distances to connect to more remote airfields or to other hubs. The algorithm is detailed in Algorithm 3-2 – Airfield Connection Algorithm.

```
Algorithm 3-2: Airfield Connection Algorithm
Data: set of airfields $N$ of size $n$, where airfield elements detail a 2-dimensional vector.
Result: a map of established airfield connections.
(1) compute the set of potential airfield connections $\Gamma$:
        $\Gamma \leftarrow (\{N_1, N_2, \dots, N_n\} \times \{N_1, N_2, \dots, N_n\}) \setminus \{(N_1, N_1), (N_2, N_2), \dots, (N_n, N_n)\}$
(2) compute the map of potential airfield bi-directional connection distances $\gamma$ s.t.:
        $\forall \{i, j\} \in \Gamma(\gamma_{i,j} = \gamma_{j,i} = |j - i|)$
        $|\gamma| = |\Gamma|$
(3) compute the average potential connection distance $\bar{\gamma}$:
        $\bar{\gamma} \leftarrow \frac{\sum \gamma_{i,j}}{|\gamma|}$
(4) compute the initial maximum connection distance $\delta$ (standard deviation of $\gamma$):
        $\delta \leftarrow \sqrt{\frac{\sum(\gamma_{i,j} - \bar{\gamma})^2}{|\gamma| - 1}}$
(5) compute the set of airfield connections $\Omega$:
        $\Omega \leftarrow \left\{\{i, j\} \in \Gamma | \gamma_{i,j} \leq \delta\right\}$
(6) compute the ordered list of airfields $\omega$ by descending connection counts s.t.:
        $\forall x \in \{1,2, \dots, n-1\}(|\{\forall\{i, j\} \in \Omega | i = \omega_x\}| \geq |\{\forall\{i, j\} \in \Omega | i = \omega_{x+1}\}|)$
(7) initialize timestep $t \leftarrow 1$
(8) do:
(9)     compute the set of airfield hubs $\Theta$:
            $\Theta \leftarrow \begin{cases} \{\omega_1, \omega_2, \dots, \omega_t\} & t \leq n \\ \omega & \text{else} \end{cases}$
(10)    compute the set of airfields unreachable from $\omega_1$ as $\Phi$:
            $\Phi \leftarrow \{i \in N \setminus \{\omega_1\} | i \text{ is not reachable from } \omega_1 \text{ via } \Omega\}$
(11)    $\delta \leftarrow \delta + t\sqrt{n\delta}$
(12)    for each airfield connection $\{i, j\} \in \{\Theta \times \Phi\} \cap \Gamma$:
(13)            if $\gamma_{i,j} \leq \delta$ then:
(14)                    $\Omega \leftarrow \Omega \cup \{i, j\}$
(15)    end for each
(16)    $t \leftarrow t + 1$
(17) while $\Phi$ is not empty
(18) end do-while
(19) return $\Omega$
```

Algorithm 3-2 – Airfield Connection Algorithm

There are a few things to note in the detailed algorithm. The connection distance variable δ—the standard deviation of the distances between all airfields γ—is selected because it was found to scale well with the count of airfields such that it typically had a few initial connections but did not initially connect all airfields. Additionally, the algorithm continues until the connection map variable Ω details connections such that any node can reach ω_1; there could be multiple aircraft hubs in Θ that were not yet connected. This allowed for when hubs are exploring the unconnected airfields to also be able to connect to other hubs, which allowed for a decentralized connection-growth process as they did not have to wait for the main hub ω_1 to connect to them and could search for their own reasonable connections. Also note that once an airfield is connected to ω_1 that it can no longer establish any new connections with the exception of any other airfields hubs that attempt to connect to it within the same timestep. This allows for multiple cycle-paths to be established between clusters or remote nodes, encouraging fuller spaces of airfield connections between what should be considered likely routes.

Finally, this is an entirely deterministic process; there is no stochastic variation in how connections are determined once the positions of the airfields are known. This was considered acceptable, as slight variations in the initial starting positions could result in significantly different connection graphs and because it is a nonlinear relation that should be fairly difficult for autonomous agents to learn to model based on airfield positions alone. That is, while the principles of the algorithm are relatively simple the steps themselves are sufficiently complex that it was found to need no further variations or added randomizations to produce airfield connections.

3.6 – Navigational Solutions

Navigational solutions are composed of the collective avoidance gradients produced by the navigable features within a shared space. Each navigational feature generates an avoidance gradient. Avoidance gradients are a function of a base strength, reach, rate of strength decay, and function of falloff, which the exact configuration then varies between navigational features. In effect, these gradients improve the simulated realism by emulating safety and adherence to regulatory policies features in flight. E.g., aircraft should not just avoid colliding with other aircraft, but should also maintain a safe distance of operation.

Base strength details the initial strength of the avoidance-factor to give to a feature—it is more important to avoid some features than others. *Reach* is the maximum range of influence of a feature, where the avoidance of spaces beyond a reach are then independent of that navigational feature. *Strength decay* determines the rate at which a feature's influence is diminished across a distance, with respect to the reach. The *function of falloff* details how to calculate the avoidance with respect to the calculated strength. One other lesser-utilized parameter is a directional reach. Most features in the scenarios have a radially uniform reach, where reach is a constant amount in every direction. *Directional reach*, then, is a reach with variable limits that are dependent on the direction from the navigational feature. This structure supports varying contours of avoidance gradients, accounting for how different navigational features may have different navigational influences and produces complex gradients used to observe a trained model's capability to simultaneously fit multiple curves in this problem space.

3.6.1 – Calculating Navigation Solutions

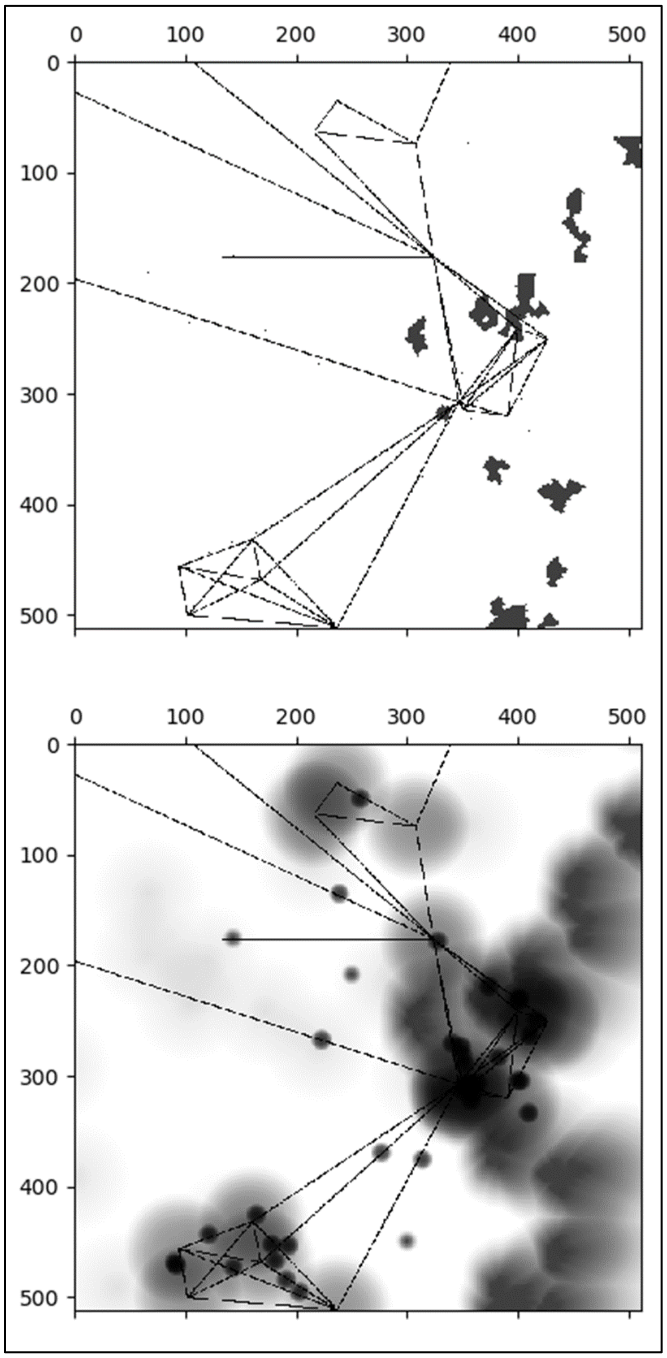

Figure 3-14 – Example Scenario (Top) and its Navigation Solution (Bottom) that Details Regions to Avoid (Shaded Regions)

Figure 3-14 details a visualized example of a scenario and its corresponding calculated navigational solution. This figure demonstrates that navigation solutions are derived from the generated scenarios, where features are given a radiating avoidance that diminishes in strength over distance. Different features have different gradients of avoidance the specifications of which are detailed in this section. Note that some avoidance gradients can be considerably weaker than others. As the navigational solutions are gray-scale images, the strength of this avoidance is given as the pixel's *value* (brightness/darkness), resulting in the darker gradients detailing higher avoidance values and brighter gradients as lower avoidance values. To better visualize some of the weaker avoidance gradients, a modified image is given in Figure 3-15 that reveals some gradients that were previously easy to miss due to their lightness.

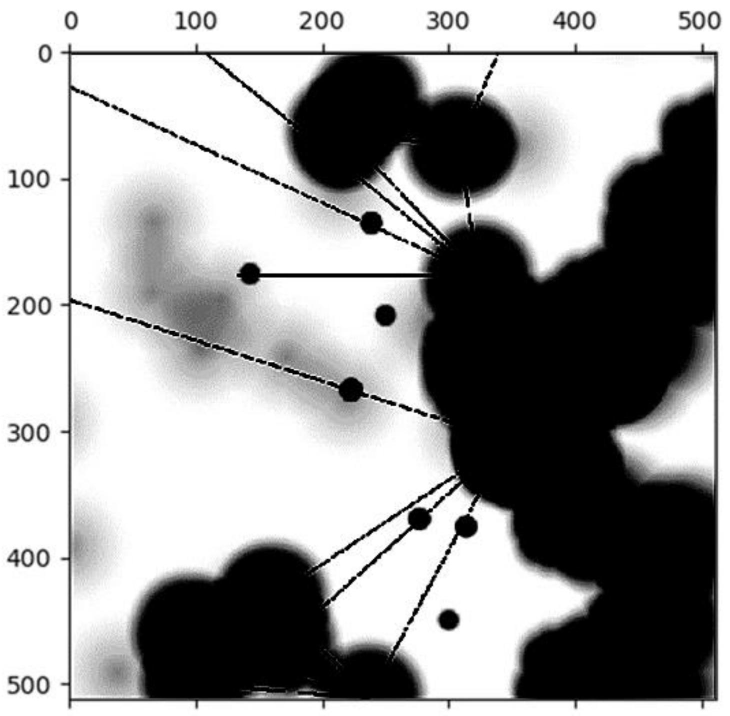

Figure 3-15 – Example Navigational Solution w/ Increased Contrast and Decreased Brightness

The previous attributes of avoidance gradients are denoted with some symbols to simplify their usage in a few expressions that demonstrate how they work: c denotes base strength, $d_{\max}$ denotes reach, β denotes the rate of strength decay, and f_x denotes some falloff function x. Note that for the reach $d_{\max}$, which can vary if the reach is directional, the expressions will be simplified as if all reaches are directionally uniform, while it is actually dependent on the vector offset of the point evaluated to produce the distance d from the navigable feature.

The symbol d is used here to denote the distance from a point in the navigable space that can be influenced by the navigable feature to the position of a navigable feature. The reference point is the coordinate of the navigable feature for point-based features. Alternatively, for polygon-based features, the reference point is the closest point along its edge. Note that in the case of directional reaches, the closest edge is always considered even if there is another edge option that could produce higher avoidance. If there is a tie between two edges, the first discovered edge-point is selected. This simplification was selected as it greatly reduces the computational complexity for evaluating these gradients. Distance constraints are given in Eq. (3.7.1-1).

$$0 \leq d \leq d_{\max} \tag{3.6.1-1}$$

The rate of strength decay details a linear strength decay at $\beta = 1$ and a constant strength value at $\beta = 0$. Rates in the range $(0,1)$ then yield a strength that decays exponentially slower and rates in the range > 1 yield a strength that decays exponentially faster. This allows further variations in the distribution of avoidance gradients given the same falloff function, emulating navigable features that can have more complex impact on the navigable solution. Constraints are given in Eq. (3.7.1-2).

$$0 \leq \beta \tag{3.6.1-2}$$

Unless otherwise specified, a default value of $\beta = 1$ will be assumed for linear behaviors.

The base strength denotes some initial priority of avoidance of the navigable feature. For polygon-based features, this also denotes the constant avoidance of the space they occupy. That is, while the domain of the avoidance values in the navigational solution is $[0,1]$, the domain of avoidance values of a navigational feature's corresponding avoidance gradient is $[0, c]$.

$$0 < c \leq 1 \tag{3.6.1-3}$$

Note that by definition of being a navigable feature, it must have some impact on the navigational solution; the strength must exceed 0.

Then, the strength of the avoidance can be determined at the given point with the function of strength $g(d, d_{\text{max}}, \beta)$. The contour of this function, with respect to the other avoidance-gradient factors, is visualized in Figure 3-16. The function of strength that returns the complement of the proportion of distance, scaled with respect to the rate of strength decay β, from an avoidance-emitting feature and the maximum reach of that feature:

$$\text{strength} = g(d, \beta) \coloneqq \begin{cases} 1 - \left(\dfrac{d}{d_{\text{max}}}\right)^{\beta} & d \leq d_{\text{max}} \\ 0 & \text{else} \end{cases} \tag{3.6.1-4}$$

There are two falloff functions implemented to add a variety of behaviors in the avoidance gradients: linear and hyperbolic tangent (tanh). Because the falloff functions are functions of strength, the linear falloff function equation is the same as the strength equation.

$$f_{\text{linear}} = g(d, \beta) \tag{3.6.1-5}$$

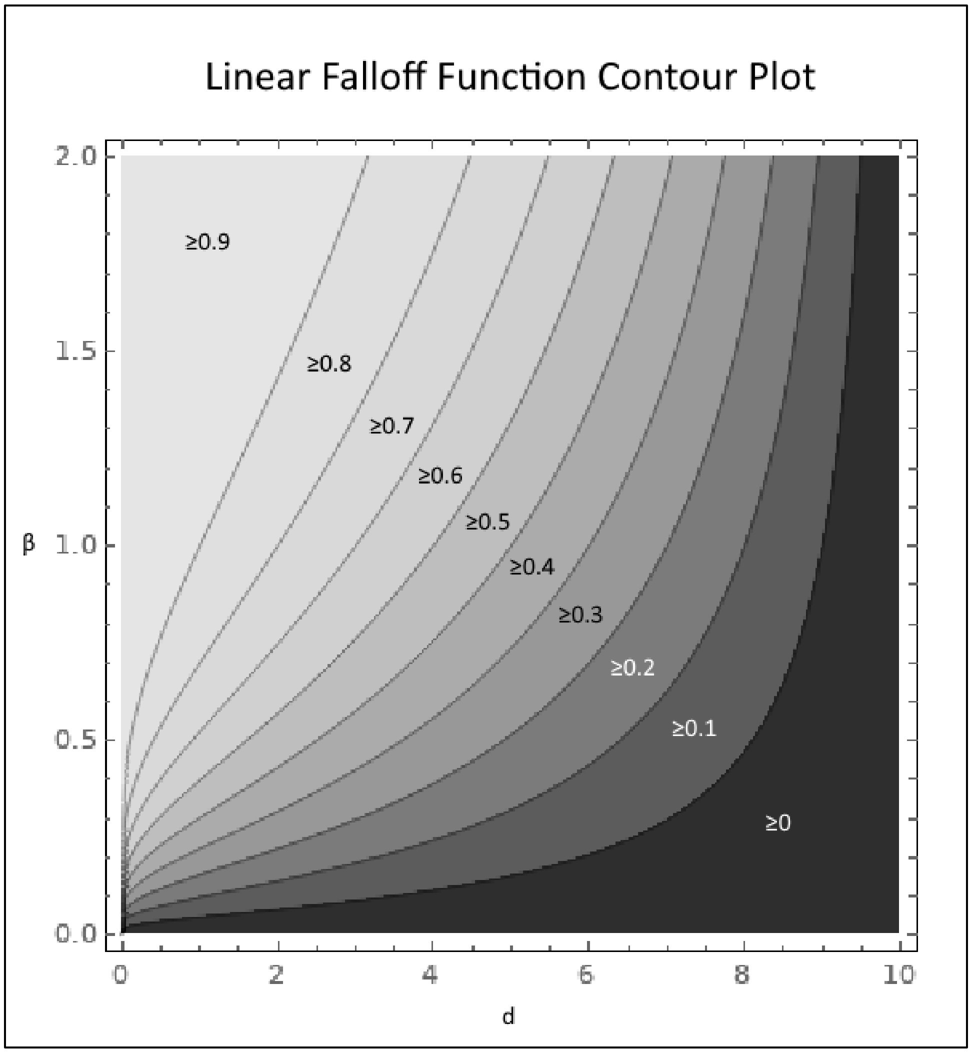

Figure 3-16 – Visualized Linear Falloff Contour (c=1, d_{max} = 10, β≤2)

Though, noting that $\beta \neq 1$ would actually indicate nonlinear behaviors in the function, transforming it into a constant or exponential function. However, this classifier is specified given the default behavior of $\beta = 1$ would detail the linear function. The tanh falloff function then provides a different gradient of avoidance falloff for contrast against the linear falloff.

$$f_{\tanh}(c, d, \beta) = c\left(\tanh\left(\frac{5.2933g(d, \beta)}{2}\right) + 0.01 * g(d, \beta)\right) \qquad (3.6.1\text{-}6)$$

Where the range of the tanh falloff function is then $[0,1)$. Note the constants in the tanh falloff function detailed in Eq. (3.7.1-6): while the upper bound of the range is not inclusive, the constant strength multiplier of 5.2933 and the offset of $0.01 * \text{strength}$ make it sufficiently close such that at 100% strength the corresponding avoidance value is approximately 0.99999995. These values were selected as the resulting distribution of avoidance gradients is sufficiently different from the linear falloff function.

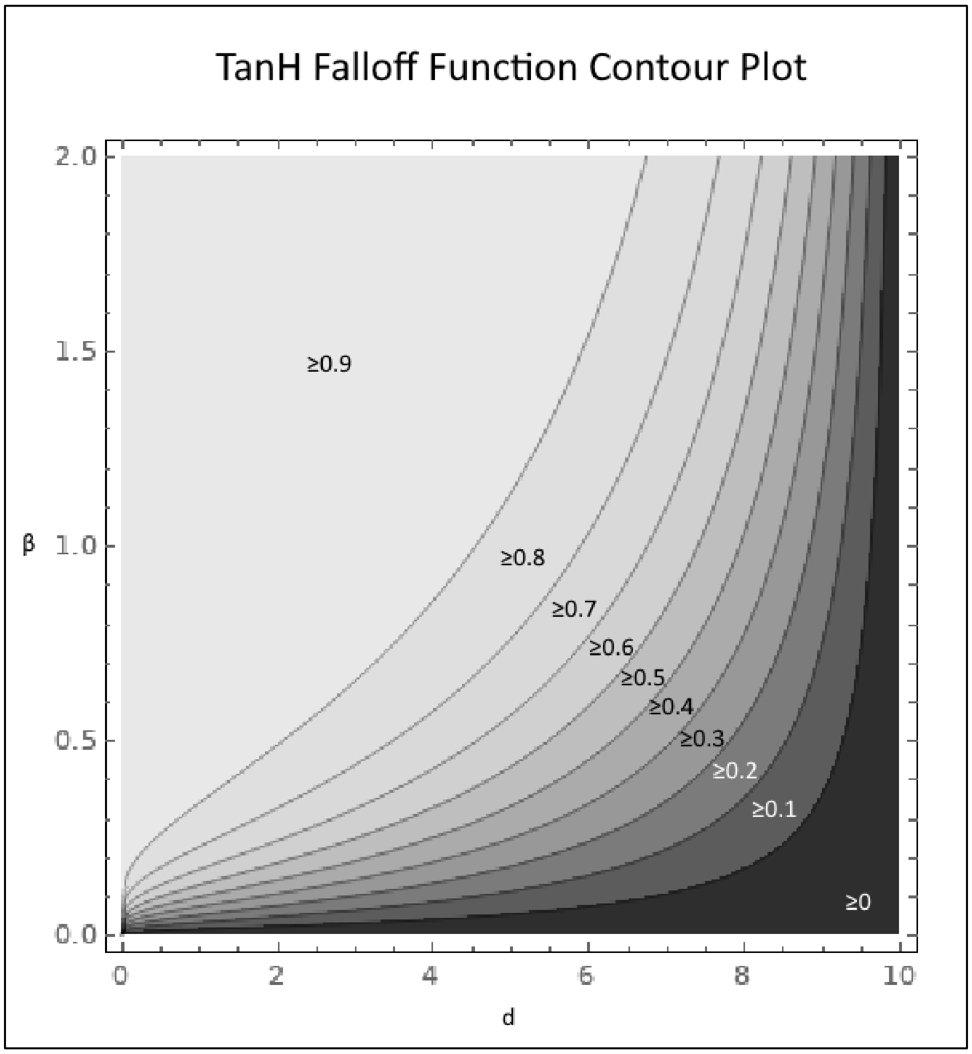

Figure 3-17 – Visualized TanH Falloff Contour (c=1, d_{max}= 10, β≤2)

3.6.2 – Navigational Solution Configurations

These configurations for the navigational solutions are selected to provide contrasting gradients of avoidance to detail variation in how the gradients of avoidances are formed. Other than that, they were designed to be locally impactful such that they could reasonably be actual avoidance gradients of real-world navigational features. For example, there is no need to account for every aircraft in a 200-kilometer radius when plotting a course. Weather fronts provide a unique distinction in that their avoidance gradient is with a directional reach maximum. This is to specifically observe in the experiments how well a non-uniform gradient can be learnt. It might also be more useful, when determining future navigable solutions, to place a greater emphasis not on where a feature exists, but to also account for directional movement of that feature.

Airfields: $f = \tanh, d_{\max} = 50, c = 0.5$

Aircraft: $f = \tanh, d_{\max} = 10, c = 0.8$

Miscellaneous Type A Features: $f = \text{linear}, d_{\max} = 50, c = 0.2$

Miscellaneous Type B Features: $f = \tanh, d_{\max} = 10, c = 0.8, \beta = 0.5$

No-Fly Zones: $f = \text{linear}, d_{\max} = 50, c = 1$

Weather Fronts: $f = \tanh, d_{\max} = [N = 37.5, S = 37.5, E = 100, W = 25], c = 0.5, \omega = 1.125$

Figure 3-18 – Avoidance Gradient Parameters

3.6.3 – Overlapping Avoidance

Points in the simulated space may have overlapping avoidance values from two or more features. Such avoidances are aggregated by calculating the product of lack of avoidances in the space and subtracting that from 100% avoidance.

$$\text{aggregate avoidance}(a, b) = 1 - (1 - a)(1 - b) \tag{3.6.3-1}$$

Where a, b are two overlapping scalar values of avoidance.

If there are greater-than 2 overlapping avoidances, the algorithm recursively performs the aggregation until all overlapping avoidances have been reduced to a singular avoidance. Note that the order of which avoidances are aggregated together does not matter. Note that if a polygon feature is generated with a fragmented geometry, all edges are considered to be part of the same feature. Therefore, there can be no multiple avoidances produced that need to be aggregated, even if a point in space is within the reach of two different points along the polygon’s collective edges.

3.7 – Simulated Aircraft Behaviors vs Learned Behaviors

The autonomous agents are being trained to identify the navigational solution of the space, yet the simulated aircraft seem to fly through storms and in extreme proximity of each other and in general seem to not adhere to any behavior that the trained autonomous agent itself should aim for. However, adding consistent behaviors would likely not have much impact on the resulting navigational solutions. The autonomous agent is being trained against the generalized environment, and is not modeling the behaviors of aircraft therein. Furthermore, it is also likely that an autonomous agent would be trained to especially avoid an area where such undesired behaviors are being exemplified, given the algorithm used to aggregate overlapping avoidances. Therefore, the seemingly contradictory scenario and navigational solutions were deemed to be sufficiently acceptable. In the future, however, when modeling more realistic aircraft behaviors—to provide increasingly realistic datasets—this might be changed.

Chapter 4 Deep Learning Architectures

DL architectures detail how the navigational features can be modeled, transformed, and trained to optimize some stochastic optimization function. Predominantly, this includes the types of neural networks (NNs) and the optimizer function used. The NNs utilized include Convolutional NNs (CNNs) to model navigational features and then either Transposed-Convolutional NNs (TCNNs) or Dense NNs (DNNs) to transform modeled navigational features into a navigational solution. This chapter further details those architectural components that are explored for the experiments and their respective properties that make them useful in this domain, as well as the Adam optimization function used to train these architectures.

4.1 – Inputs and Outputs

The shape of the architectural inputs is an ordered $\mathrm{H}\times\mathrm{W}\times\mathrm{C}$ dimensional array, where those dimensions denote the height (H) and width (W) of the input space with a channel-depth (C) where the values—the presence indicator flags—are stored. While the channel-depth is a constant length given a constant count of feature-types, height and width can be varied. There are then two types of input architectures: full-sized or sliced. Full-sized architectures—architectures that can generate a comprehensive navigational solution of the space—are the default, unless otherwise specified, where the height and width input dimensions are the same as the captured spaced, given a 1:1 resolution: 512×512. Sliced architectures—architectures that generate sliced navigational solutions that can be tiled to create the comprehensive navigational solution—then train over some reduced measures of the space, which sacrifices information for faster training times and reduced model parameters. As the architectural inputs detail the navigational features, the architectural outputs then detail the navigational solution. The shape of the output is then an ordered H×W dimensional array, where the values—the avoidance weights—are then stored along the width-axis of the array. The dimensional measures of these height and width components are dependent on the dimensions of the inputs. Given that the output resolution is also structured at a 1:1 ratio, then the height and width of the input is then the height and width of the output.

4.2 – Architectural Layout

First and foremost, architectures are selected for their compatibility and feasibility to manipulate this dimensionality and structure of data. Compatibility is a hard requirement, as otherwise the architecture would be non-functional as the input or output data would be impossible to model or produce. While feasibility is soft requirement, it effectively determines what approaches could be explored as some architectural styles would have a large computational complexity, resulting in extreme training times that make it not practical to implement. As navigational features are linked to an avoidance gradient, which is a positional distribution of values about the navigational features space, the relations between the input and output have spatial connotations. Therefore, of the explored architectures there is also an emphasis on selecting architectures that can model these types of spatial relations.

Given the input shape details a 3-dimensional array of spatial information, it is identical to the structure of an image. CNN architectures are then selected to model this information, as they are effectively used in the domain of image processing (Krizhevsky, et al., 2012). CNNs input multi-dimensional data and produce multi-dimensional information that is potentially of different sizes and measures. This information is denoted as a *feature map* and is produced via convolving the input dimensional data with some trained weights.

It is important to know, if unfamiliar with convolutions, that convolutions typically reduce the measures along the spatial-axes. Thus, it is required that transforming NNs be able to increase these measures to match the required size of the output. Then, the architectural approaches that are used to transform this output information must also be compatible with multi-dimensional inputs, be able to model spatial properties, and detail an output with measures greater than the input. TCNNs are then a viable candidate, as they, per the name, perform transposed convolutions which can transform a feature map of some convolution to its original input (Dumoulin and Visin, 2018). Alternatively, to TCNNs are the traditional DNNs, which intake some multi-dimensional vector and output another multi-dimensional vector by multiplying the input vector by a matrix of weights. Note that while a CNN outputs a multi-dimensional array, any array can be flattened into a multi-dimensional vector, and vice versa. Therefore, with these additional reshaping steps, a DNN is compatible within the architecture.

Two main architectures are yielded from this selection of components: Conv-to-ConvT and Conv-to-Dense architectures. These labels do not explicitly mean that a Conv-to-ConvT architecture details a single CNN layer and then a single TCNN layer; these architectures detail some number of CNN layers used to construct the modeled navigational feature information and then some number of TCNN layers to then transform the information into the navigational solution. The same is true for the Conv-to-Dense architectures. There are thus infinitely many realizations of these architectures, where the experiments then detail the best discovered realizations. As previously mentioned, the inputs to these architectures default to the entire dimensional measures of the discretized captured space. There are then specific architectural variants that vary not by their layout but instead by inputting a slice of the discretized space. This is used to yield the Sliced Conv-to-Dense Architecture, as the main benefit is reducing model parameters and DNNs yield exponential model parameter counts—mainly the number of weights—whereas TCNNs yield constant model parameter counts, with respect to the shapes of the inputs and outputs.

The flow of information in these architectures is sequential; both architectures form what are known as sequential models. *Sequential models* entail that the comprehensive input for each layer is: 1) the original model input for the first layer in the model or 2) otherwise the collective output of the previous layer for all remaining layers. Thus, these architectures are self-contained and are then only dependent on the initial model input.

4.3 – Architectural Component Summaries

For readers unfamiliar with the explored architectural components—CNNs, TCNNs, and DNNs—their mechanical summaries are provided in following subsections.

4.3.1 – Convolutional Neural Networks

CNNs map some n-dimensional array to some m-dimensional array, where m is the number of filters used in the CNN (e.g., Goodfellow, et al., 2016). Filters are some $n-1$ dimensional array that map segments of the input to an output known as a feature map. These mappings are produced by a dot-product between the filter and a slice of the input with the same dimensional measures as the filter, along the last axis—the channels—of the input n-dimensional array, which produces an $n - 1$ dimensional output that is aggregated vis summation into a scalar value. Each filter slides over the input data, per some step size, mapping the resulting scalars to some corresponding step-based index of the feature map.

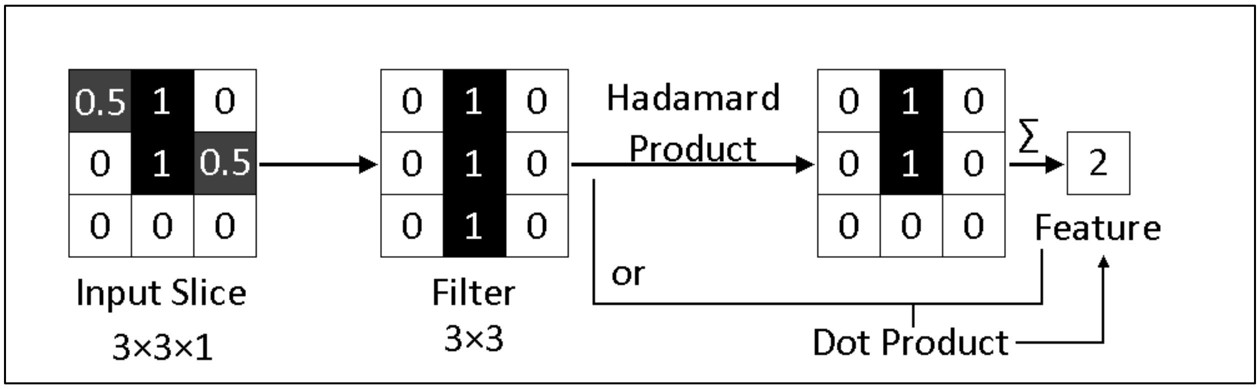

Figure 4-1 – Convolution Example

CNNs have a number of parameters used to configure the properties of the layer. Two of the relevant parameters are kernel and step sizes, as variations of these parameters are detailed in the specifications of the demonstrated architectures. The dimensional measures of a filter--*kernel sizes*—are typically uniform along each axis. The sliding window of the filter is incrementally offset by a step (aka *stride*) size, which is similarly typically uniform along each axis. An example of how a series of convolutions is transformed into a feature map with a step size of 2×2 is detailed in Figure 4-3. Note the numbers in correlate the output convolutions to their respective positions in the resultant feature map. Note filters emulate the synapse strengths in the CNN. They receive input signals from an input-slice and produce some corresponding output neuron. How these synapses are formed is a result of the selected optimizer fitting the weights to minimize/maximize some objective function. Because the kernel size and filter count are structurally independent of the input and output shapes, with the exception that the kernel sizes must be smaller than the input shape, then the weights of a CNN are constant. Thus, CNNs scale well with respect to the input and output sizes. CNNs typically detail multiple filters, where more filters increase the capacity for the number of features in the targeted input that can be identified.

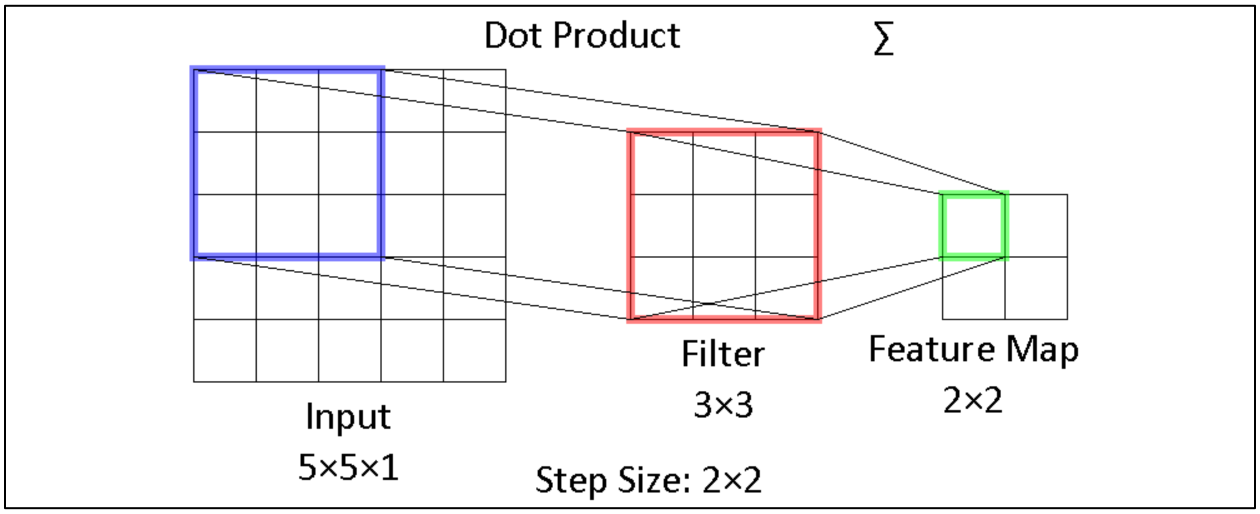

Figure 4-2 - CNN Convolution Visualization

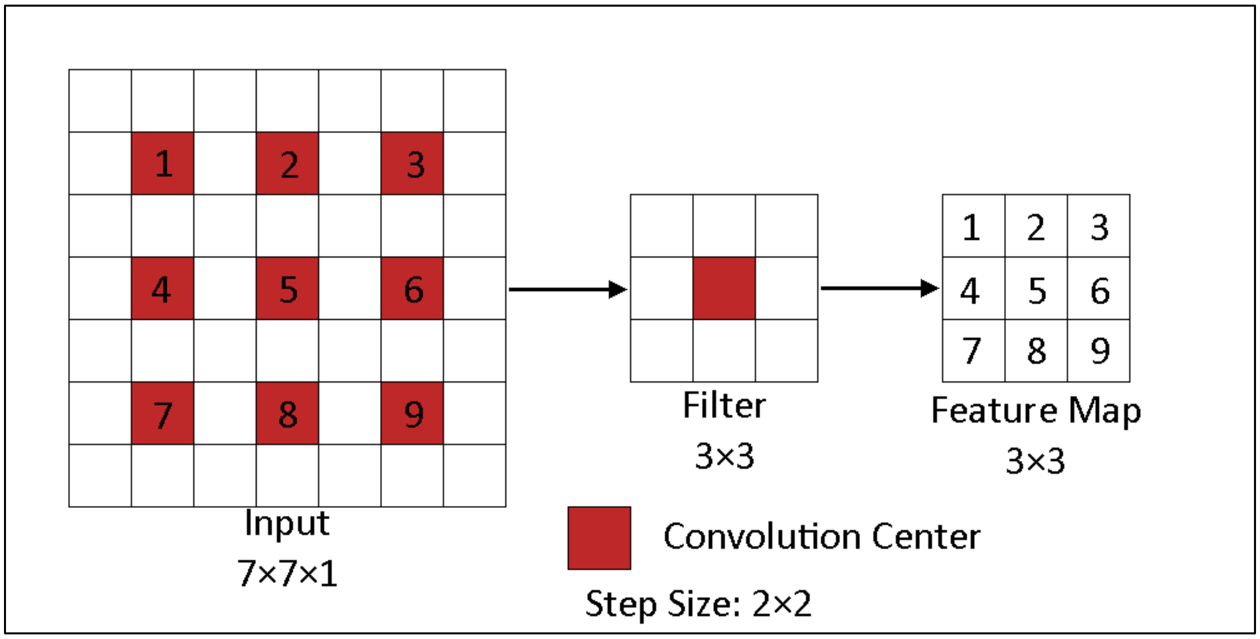

Figure 4-3 – CNN Full Visualization

4.3.2 – Transposed-Convolution Neural Networks

TCNNS are largely identical to CNNs and can be considered a reversal to an extent. TCNNs take in an m-dimensional array and output an n-dimensional array, where n is the number of filters. Similarly, filters are some $m-1$ dimensional array used to map each individual features of the input to the output. These mappings are detail a linearly scaling of a filter by these scalar features. Consider the output of a TCNN to initially be some zero n-dimensional array, then each linearly scaled filter is added to the values in the output array in the same iterative fashion as a CNN builds its feature map. Because of their similarity to CNNs, many of their behaviors and parameters are shared.

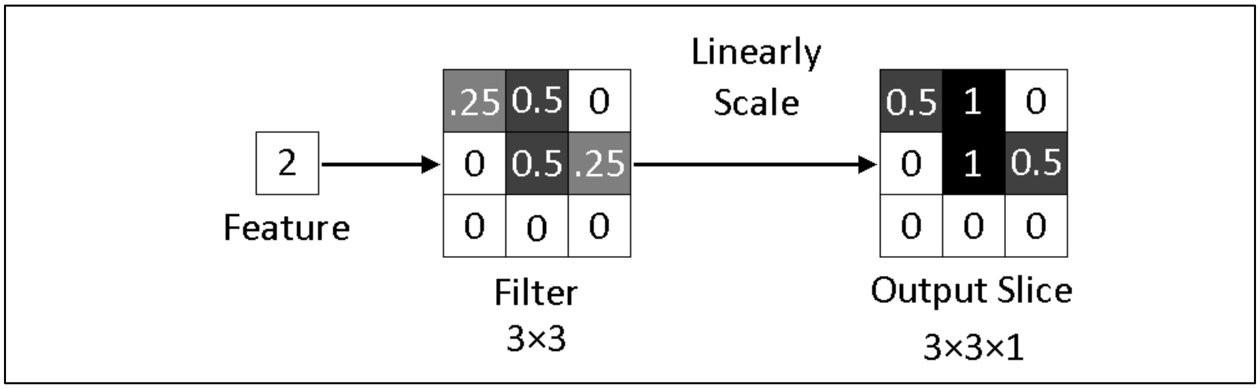

Figure 4-4 – Transposed-Convolution Example

TCNNs have a number of similar parameters to those of a CNN. However, the step size instead now details how the linearly scaled filters are summed across the output. An example of a transposed-convolution is given in Figure 4-4 with a simplified view of the TCNN detailed in Figure 4-6 with the partial equations for how to calculate the output detailed. Weights are similarly independent of input and output dimensional measures.

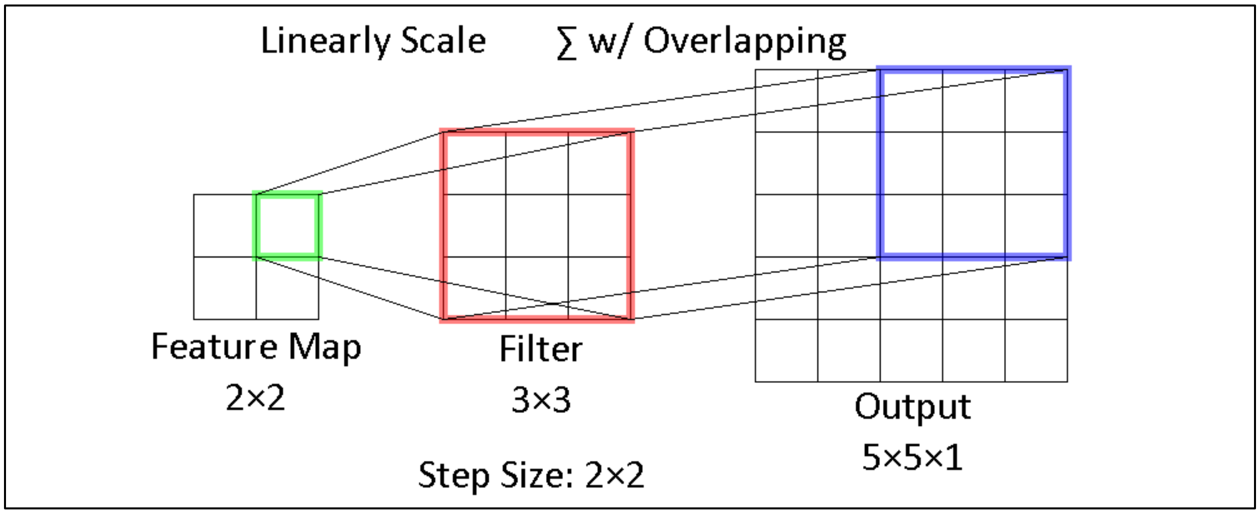

Figure 4-5 - TCNN Transposed-Convolution Visualization

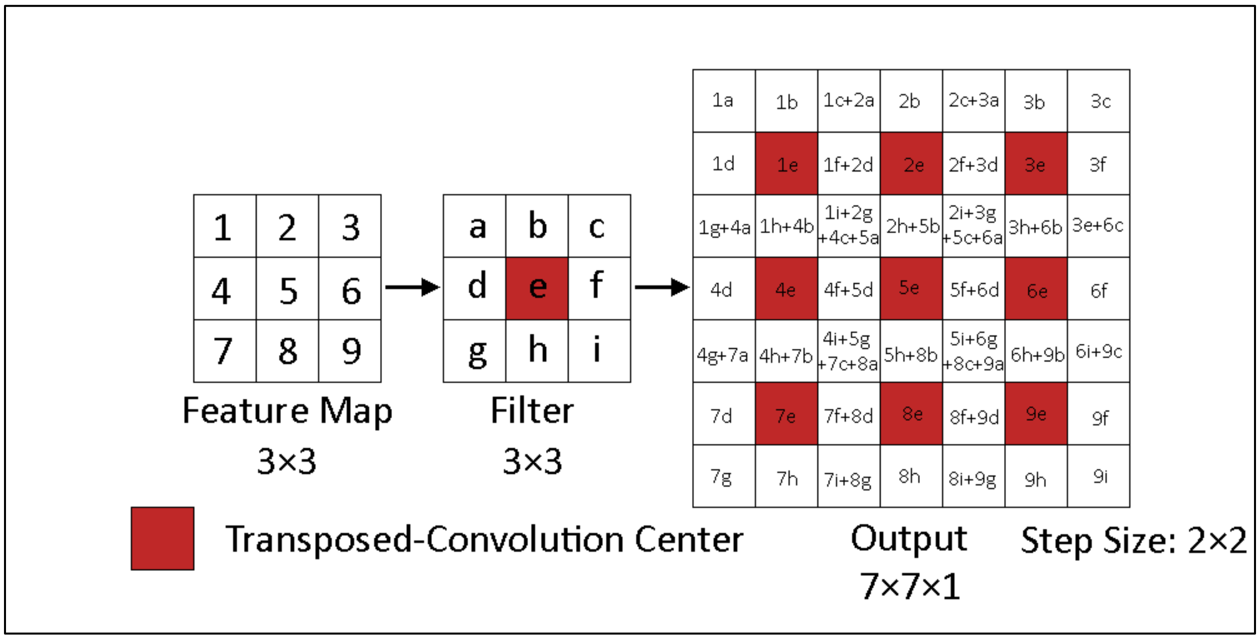
1 2 3
4 5 6
7 8 9
Feature Map
3×3
a b c
d e f
g h i
Filter
3×3
1a 1b 1c+2a 2b 2c+3a 3b 3c
1d 1e 1f+2d 2e 2f+3d 3e 3f
1g+4a 1h+4b 1i+2g +4c+5a 2h+5b 2i+3g +5c+6a 3h+6b 3e+6c
4d 4e 4f+5d 5e 5f+6d 6e 6f
4g+7a 4h+7b 4i+5g +7c+8a 5h+8b 5i+6g +8c+9a 6h+9b 6i+9c
7d 7e 7f+8d 8e 8f+9d 9e 9f
7g 7h 7i+8g 8h 8i+9g 9h 9i
Transposed-Convolution Center
Output
7×7×1
Step Size: 2×2

Figure 4-6 – TCNN Full Visualization

4.3.3 – Dense Neural Networks

DNNs map some input n-dimensional vector to some output m-dimensional vector (e.g., Goodfellow, et al., 2016). As previously mentioned, these are still compatible with the flow of the architecture as multi-dimensional inputs can be flattened into a vector and vectorized outputs can be reshaped into a multi-dimensional output, where the product of dimensional measures of the output equals the length of the output vector. This is a very simple structure that can be simply notated as

$$\vec{z}^{\mathrm{T}} W_{i\times j} = \vec{o} \tag{4.3.3-1}$$

$$|\vec{z}| = i \tag{4.3.3-2}$$

$$|\vec{o}| = j \tag{4.3.3-3}$$

Where $\vec{z}^{\mathrm{T}}$ is some transposed vector of i input neurons, $W_{i\times j}$ is some matrix of weights of shape $i \times j$, and $\vec{o}$ is then the vector of j output neurons.

While TCNNs typically only partially connect some of the input to the output, assuming kernel sizes less than the input sizes, DNNs fully connect all inputs to all outputs. This dense connection of weights is both a benefit and detriment, as it allows long-distance features to be correlated. This is in contrast to TCNNs that only correlate local features. Observe Figure 4-6 that details some examples on how the output signals are formed; per each filter, a single output value can, at most, be impacted by up to 4 features. DNNs would, in comparison, allow for all output values can be impacted by all input features. While this can be an effective modeling tool, this comes at a significant cost: an exponential count of model weights. Given that the definition of weights provided in Eq. (4.2.3-1) details the matrix of weights being dependent on the input and output shape; for every output neuron a column of weights is added to W and for every input neuron a row of weights is added to W. Therefore, its usage is especially limited when dealing with large amounts of data. One strategy is pooling the input convolutions to reduced parameter counts to make the model more feasible, but this adds some loss to the information. However, it is explored here with the Conv-to-Dense architectures given the otherwise infeasibility task of training the model.

4.2 – Model Optimizer

The loss measured between the actual and predicted navigational solutions can be used to adjust the weights per the selected optimizer. Therefore, the selection of an optimizer is an important process and an integral component of the architecture. For these experiments, the Adaptive Moment Estimation (Adam) optimizer is universally used. The features of this optimizer include spatial efficiency, scales well with large parameter sizes, and the generalized capability to quickly optimize models that compute a stochastic function (Kingma and Ba, 2015). The Adam optimizer is also demonstrated to converge faster than other optimization functions, such as AdaGrad, RMSProp, SGDNesterov, and AdaDelta, when training a sequential model of CNNs (ibid). Because all the explored architectures are dependent on CNNs to develop the feature maps, the Adam optimizer was deemed suitable.

Chapter 5 *n*-Array Balanced Type Error Measures

As the problem space details regression modeling, a regression loss function needs to be selected for training the developed architectural models. However, traditional regression loss functions do not satisfy the requirements of both: accounting for the positional distributions of avoidance values across the space and accounting for the bias in the aggregate total of avoidance values with respect to the entire space; the summation of avoidance values across the scenarios are not equivalent to the complementary lack of avoidance values. Therefore, a new type of regression loss function is proposed: the *n-Array Balanced Type Error* (NBTE) Loss Function, which simultaneously equalizes the loss penalties from Type I and II Errors (false positives and negatives, respectively).

The NBTE function operates by a pointwise comparison between the actual and predicted values—two *n*-dimensional arrays—where the magnitudes of each corresponding error types are aggregated. This means that it is not sufficient to learn the same amounts of avoidances to predict; to reduce the loss, the avoidance gradients must be positioned correctly as well as detailing the correct amounts of avoidance. Scaling factors are computed to determine how to balance the realized amounts of Type I and II Error and these balanced errors detail the loss. One final step is a normalization process that applies a linear scaling to these error losses to bind the error to the continuous range of $[0,1]$. The algorithm is explicitly detailed Algorithm 5-1 – NBTE Loss Algorithm.

Because this output range is normalized, the complement then provides a metric for evaluating the accuracy of the model: the NBTE Accuracy. However, the accuracy metric does not directly indicate the quality of the solution; a high accuracy model is not as impressive if a random prediction is also expected to yield an equally high level of accuracy. An additional metric to then measure the quality of information is also introduced: NBTE Coherence. This will evaluate the accuracy of a prediction with respect to the mean accuracy of a randomized prediction to describe whether or not the model can provide meaningful information.

```
**Algorithm**: $n$-Array Balanced Type Error (NBTE) Loss Function $\vartheta_L$
**Data**: Two $n$-dimensional arrays for the actual value $y$ and the predicted value $y'$ with scalar values in the domain $[0, z]$ that both share the same dimensionality $d$.
**Result**: A continuous, real scalar in the range [0,1] detailing a loss with balanced, linear penalties from Type I and II Errors.
(1) Compute the size of the space of $y$ and $y'$ as $s$:
        $s \leftarrow \prod_{i=1}^{n} d_i$
(2) Compute the weighting of Type I Error as $a$:
        $a \leftarrow \begin{cases} \frac{zs}{\sum(z-y)} & 0 < \sum(z-y) \\ zs & \text{else} \end{cases}$
(3) Compute the weighting of Type II Error as $b$:
        $b \leftarrow \begin{cases} \frac{zs}{\sum y} & 0 < \sum y \\ zs & \text{else} \end{cases}$
(4) Compute the amount of Type I Error as $\tau$:
        $\tau \leftarrow \begin{cases} \sum \min(\max(y' - y, 0), z) & 0 < \sum(z-y) \\ 1 & \text{else} \end{cases}$
(5) Compute the amount of Type II Error as $\upsilon$:
        $\upsilon \leftarrow \begin{cases} \sum \min(\max(y - y', 0), z) & 0 < \sum y \\ 1 & \text{else} \end{cases}$
(6) Compute the measure of NBTE Loss:
        $\text{loss} \leftarrow \frac{a\tau + b\upsilon}{2zs}$
(7) **return** loss
```

Algorithm 5-1 – NBTE Loss Algorithm

Where the notation to calculate the loss of some prediction y' with respect to the actual solution y is then:

$$\text{nbte loss} = \vartheta_L(y, y') \tag{5-1}$$

Where the NBTE Accuracy is given by the function ϑ_A is then calculated as:

$$\text{nbte accuracy} = \vartheta_A(y, y') \coloneqq 1 - \vartheta_L(y, y') \tag{5-2}$$

5.1 – Normalization Proof

To normalize the losses incurred by the errors, their sums are linearly reduced by a factor of $2zs$. The proof of which follows.

Claim: $a\tau + bv \leq 2zs$

Given the definitions and constraints of the variables defined in Algorithm 5-1 – NBTE Loss Algorithm, then there 3 cases to consider:

1. there is no potential for Type I Error,
2. there is no potential for Type II Error, and
3. there is potential for both Type I and II Error:

Case 1) There is No Potential for Type I Error

Proof by substitution:

For there to be no potential Type I Error, the summation of the elementwise complements of values in the n-dimensional array must be zeroed. Then, any prediction in the domain of the n-dimensional array, the real continuous interval $[0, z]$, will be less than or equal to the actual value; Type II Error is possible, but not Type I Error. These constraints, and the conditional determinations of a, b, τ, v follow are then given as:

$$\sum(z - y) = 0 \tag{5.1-1}$$

$$a = zs, b = \frac{zs}{\sum y} \tag{5.1-2}$$

$$\tau = 1, v = \sum \min(\max(y - y', 0), z) \tag{5.1-3}$$

Assuming maximum realized Type II Error, then the loss is calculated as:

$$v = \sum y \tag{5.1-4}$$

$$a\tau + bv = (zs)(1) + \frac{zs}{\sum y}\sum y = 2zs \tag{5.1-5}$$

$\therefore$ for Case 1, $a\tau + bv \leq 2zs$

Case 2) There is No Potential for Type II Error

Proof by substitution:

For there to be no potential Type II Error:

$$\sum y = 0 \tag{5.1-6}$$

$$a = \frac{zs}{\sum(z-y)}, b = zs \tag{5.1-7}$$

$$\tau = \sum \min(\max(y' - y, 0), z), v = 1 \tag{5.1-8}$$

Assuming maximum realized Type I Error, then the loss is calculated as:

$$\tau = \sum(z-y) \tag{5.1-9}$$

$$a\tau + bv = \frac{zs}{\sum(z-y)}\sum(z-y) + (zs)(1) = zs + zs = 2zs \tag{5.1-10}$$

∴ for Case 2, $a\tau + bv \leq 2zs$

Case 3) There is Potential for Both Type I and II Error

Given that there is potential for both Type I and II Error, then the following constraints are true:

$$\sum \min(\max(y' - y, 0), z) \leq \sum(z-s) < zs \tag{5.1-11}$$

$$\sum \min(\max(y - y', 0), z) \leq \sum y < zs \tag{5.1-12}$$

Which is to say, the realized Type I and II error cannot exceed the maximum potential, which is less than zs. Let α, β denote the respective proportion of realized Type I and II Error to the maximum potentials.

$$\alpha = \frac{\sum \min(\max(y' - y, 0), z)}{\sum(z-y)} \tag{5.1-13}$$

$$\beta = \frac{\sum \min(\max(y - y', 0), z)}{\sum(y)} \qquad (5.1\text{-}14)$$

Then, the following constraints are given:

$$0 < \alpha < 1 \qquad (5.1\text{-}15)$$

$$0 < \beta < 1 \qquad (5.1\text{-}16)$$

Because both the potential for Type I and II error is less than zs, the following constraint will be proved: $a\tau + bv < 2zs$; the non-normalized loss approaches $2zs$. Which is given to still satisfy the original claim.

Proof by substitution:

$$a\tau + bv < 2zs \qquad (5.1\text{-}17)$$

$$\Rightarrow zs\alpha + zs\beta < 2zs \qquad (5.1\text{-}18)$$

$$\Rightarrow zs(\alpha + \beta) < 2zs \qquad (5.1\text{-}19)$$

$$\Rightarrow a + \beta < 2 \qquad (5.1\text{-}20)$$

∴ for Case 3, $a\tau + bv < 2zs$

∴ The constraint $a\tau + bv \leq 2zs$ holds true for all three cases and the claim is then accepted.

Q.E.D.

5.2 – NBTE Loss and Accuracy Example

The example detailed in this section demonstrates some prediction that is as inaccurate as possible. However, the neither actual n-dimensional array y with dimensionality d is not the identity matrix scaled by the maximum value z nor is it the zero matrix:

$$y \neq zI_d \tag{5.2-1}$$

$$y \neq 0_d \tag{5.2-2}$$

Therefore, the loss can only approach 1 as there is a minimum degree of correctness for the values not at the endpoints of the domain $[0, z]$.

Example:

Given the definitions of n, d, z, y, y':

$$n = 2 \tag{5.2-3}$$

$$d = (3,3) \tag{5.2-4}$$

$$z = 1 \tag{5.2-5}$$

$$y = \begin{bmatrix} 0 & 0.25 & 0.5 \\ 0.25 & 1 & 0.25 \\ 0.5 & 0.25 & 0 \end{bmatrix} \tag{5.2-6}$$

$$y' = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \end{bmatrix} \tag{5.2-7}$$

Then, the loss is calculated with 4 steps derived from the algorithm detailed in Algorithm 5-1 – NBTE Loss Algorithm:

1. calculating the measure of the space,
2. calculating the weighting to equalize the penalties of Type I & II Errors,
3. calculate the realized amounts of Type 1 & II Errors, and
4. compute the measure of NBTE loss.

Additionally, with the NBTE loss known, the corresponding metric of accuracy, NBTE accuracy, can be computed.

Step 1 – Calculate the Measure of the Space s

$$s = \prod_{i=1}^{|n|} d_i = (3)(3) = 9 \tag{5.2-8}$$

Step 2 – Calculate Weights to Equalized Type I and II Error Penalties

$$a = \frac{zs}{\sum(1-y)} = \frac{9}{6} = 1.5 \tag{5.2-9}$$

$$b = \frac{zs}{\sum y} = \frac{9}{3} = 3 \tag{5.2-10}$$

Step 3 – Calculate the Amounts of Type I and II Error

$$0 < \sum(1-y) \rightarrow \tau = \sum \min(\max(y' - y, 0), z) = \sum \begin{bmatrix} 1 & 0.75 & 0.5 \\ 0.75 & 0 & 0.75 \\ 0.5 & 0.75 & 1 \end{bmatrix} = 6 \tag{5.2-11}$$

$$0 < \sum y \rightarrow \upsilon = \sum \min(\max(y - y', 0), z) = \sum \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} = 1 \quad (5.2\text{-}12)$$

Step 4 – Calculate the NBTE Loss

$$\text{nbte loss} = \vartheta_L(y, y') \coloneqq \frac{a\tau + b\upsilon}{2zm} = \frac{(1.5)(6) + (3)(1)}{2(1)(9)} = \frac{2}{3} = 0.6\bar{6} \quad (5.2\text{-}13)$$

Computing the NBTE Accuracy:

$$\text{nbte accuracy} = \vartheta_A(y, y') \coloneqq 1 - \vartheta_L(y, y') = 1 - 0.6\bar{6} = 0.3\bar{3} \quad (5.2\text{-}14)$$

5.3 – Neutral Loss and Accuracy and NBTE Coherence

While the NBTE Accuracy measure details the accuracy of a prediction, it is not inherently evident of the quality of the prediction; some realized accuracy of a prediction is not very meaningful if the expected accuracy of a random prediction is approximately equivalently accurate and precise. Therefore, the term *Neutral Accuracy* is introduced, which is defined as the frame of reference with which the NBTE Accuracy of some prediction can be compared against to evaluate the quality of the prediction. The Neutral Accuracy is the expected NBTE Accuracy of some realization of the random distribution $\mathbb{U}(0, z)^{n \times n}$ with respect to the distribution of the problem space Y, denoted as the function $\vartheta_E(Y)$:

$$\text{neutral accuracy} = \vartheta_E(Y) \coloneqq \mathrm{E}_Y[\sim\mathbb{U}(0, z)^{n \times n}] \quad (5.3\text{-}1)$$

$$\text{neutral loss} = 1 - \vartheta_E(Y) \quad (5.3\text{-}2)$$

The uniform distribution is selected here as it has no specific bias of the values selected. Thus, it provides an unbiased frame of reference for other predictions. The meaningfulness of a prediction—the coherence—is then measured with respect to both the Neutral Accuracy. This measure is referred to as the NBTE Coherence metric, which is denoted as the function $\vartheta_C(y, y', Y)$:

$$\text{nbte coherence} = \vartheta_C(y, y', Y) \coloneqq \frac{\vartheta_A(y, y') - \vartheta_E(Y)}{1 - \vartheta_E(Y)} \tag{5.3-3}$$

The range of the NBTE Coherence is then given as $(-\infty, 1]$ with the associated constraints. Because $\vartheta_C(y, y') = 1$ indicates maximum prediction and $\vartheta_C(y, y') = 0$ indicates then minimum quality, then $\vartheta_C(y, y') < 0$ then correlate to some potential that the complement of a prediction, with respect to the upper-bound of the domain, will be more accurate than the original. Because a complementary prediction has chance to introduce more error along an axis than the original, the accuracy of the complement is not guaranteed to be the complement of the original prediction's accuracy. The point at which the complementary prediction would become more reliable is conjectured to lie somewhere in the domain $(-\infty, 0)$. However, such properties are not explored as it is not immediately required for the experiments detailed herein. It is typical for the NBTE Coherence to be in the domain $[0,1]$, which indicates that the minimum expected accuracy is the Neutral Accuracy score, as a model should typically be able to construct a random guess at worse. The relation between the NBTE Accuracy and the Neutral Accuracy is detailed in a contour plot in Figure 5-1.

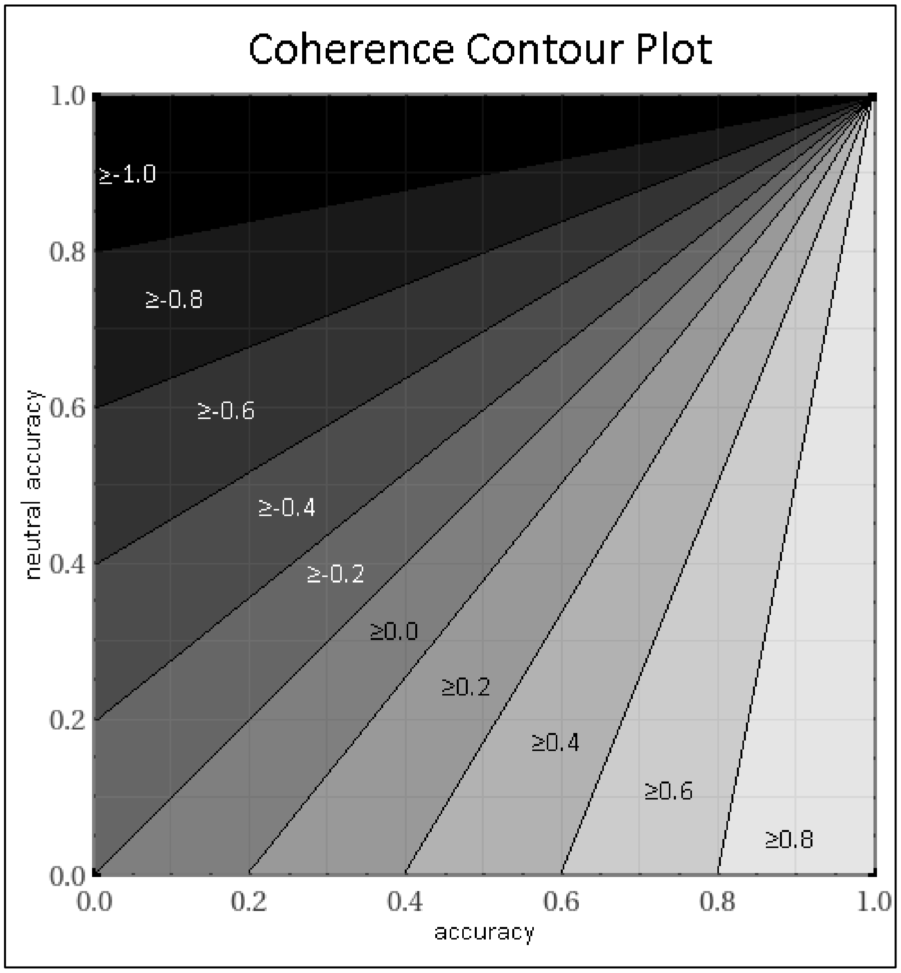

Figure 5-1 – NBTE Coherence Contour Plot

The Neutral Accuracy is empirically determined by sampling the target distribution greater-than-or-equal to 30 times such that the sampled means and standard deviations approximate the Normal distribution per the Central Limit Theorem. The dependency of the Neutral Accuracy on the target distribution is visualized in Figure 5-2. In those figures, contrasting Neutral Accuracy values are observed within similar distributions, detailing a further dependance on the dimensional measures of the data and the parameters of the distribution.

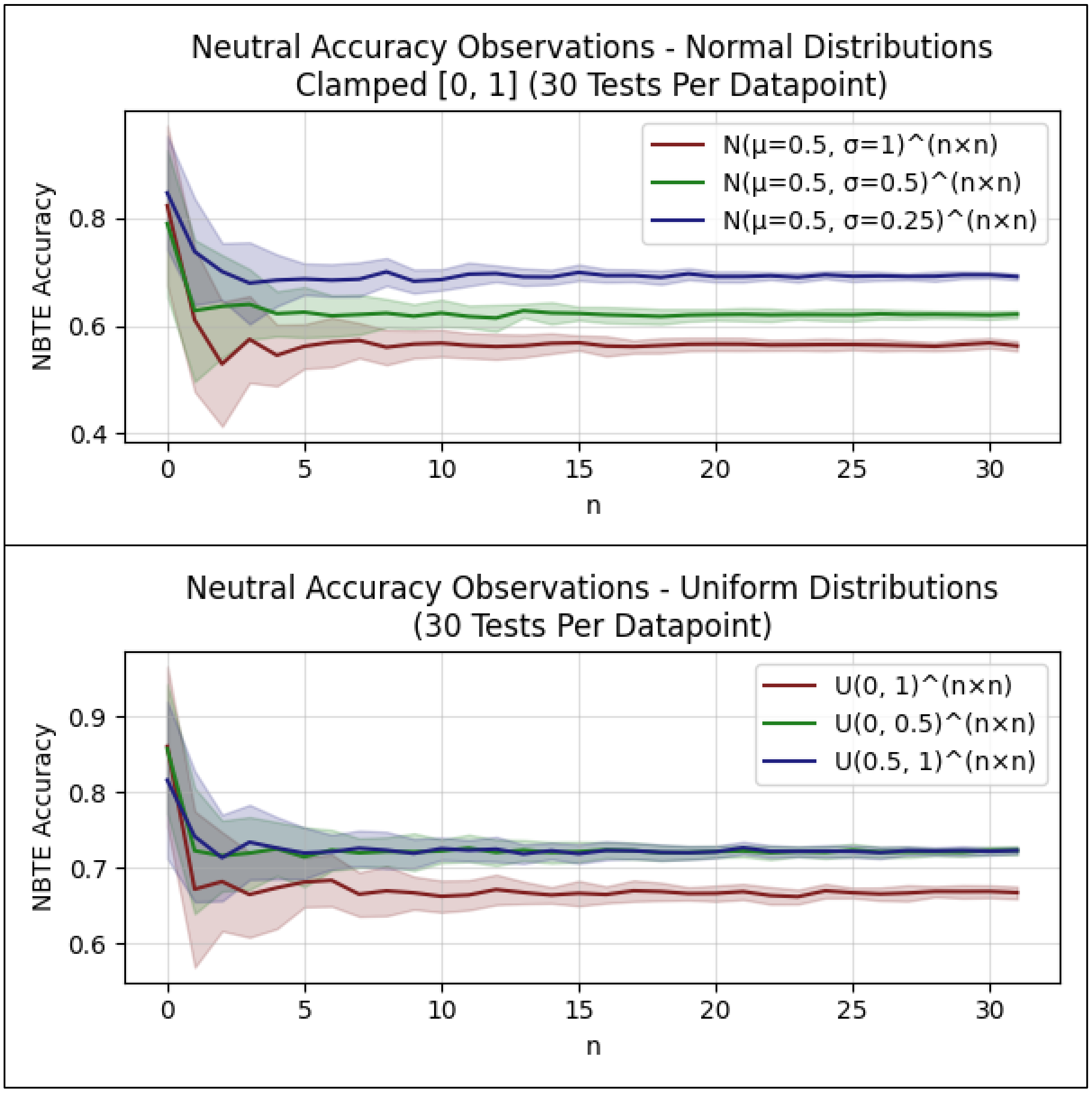

Figure 5-2 – Neutral Accuracies w/ ± Avg. Standard Deviation

Chapter 6 Experimental Setup

This chapter is a prelude to Chapter 7 – Experiments and Results, which identifies the relevant specifications of the environment used to run and evaluate the experiments. Environment specifications include: 1) dataset sizes, 2) training, validation, and testing partition sizes, 3) data sampling methodologies, and 4) commonly used model configurations and callback functions. These details should be broadly assumed to be universally applied to all the architectural models, unless otherwise explicitly specified. The counts and settings of parameters given are selected as such, due to observed successes when training the initial models.

6.1 – Dataset, Partition Sizes, and Sampling

From the previously described specifications of the scenarios, a count of 1000 discretized scenarios were procedurally generated, along with their corresponding navigational solutions. These scenarios and solutions then detail the comprehensive datasets used as model inputs. This data was partitioned into training, validation, and testing sets with proportions 60%, 20%, and 20%, respectively. Because scenarios are procedurally generated, shuffling the data before portioning it was not needed.

Models are fitted after each step in each generation. A step consists of a batch of inputs being processed by the model, where the averaged loss of the batch is processed by the optimizer and adjusts the model weights accordingly. Batches are used in a step to reduce the impact outliers would have on the data, as their outlying value is diminished by the averages. However, too many items in a batch could impede learning rates as it takes time to update the model weights and observing too many values can be redundant as the averaged loss is not likely to change by a significant margin after some sufficient number of inputs. Inputs are randomly grouped in batches of 16. If there are leftovers, they are not considered for the current generation due to a technical limitation that, when being trained, a model is configured to expect 16 items and cannot support a different count other than that. However, unused inputs in one generation can potentially be used in the next, as which inputs selected is randomly determined every generation. Note that for sliced architectures, they observe a random slice of the input scenarios, which is tested against the corresponding slice in the navigational solution.

6.2 – Model Callbacks

Model callbacks allow for conditional actions to be taken at the end of each generation of training, typically dependent on the state of the model's hyperparameters, such as current loss for the training or validation data, learning rates, the metrics used to provide a quick evaluation of the model at its current state, and other stateful properties. In this experiment, three callbacks are used: Checkpoints, Reduce Learning-Rate on Plateau + Walk-Back, and Early Stopping.

6.2.1 – Callback – Checkpoints

As the name implies, the *Checkpoint* callback creates a checkpoint of the current model weights. This allows for restoring the model weights between concurrent sessions of training and for loading the weights after training is completed to make predictions. For this experiment, a new checkpoint is made anytime there is a record-low value of the validation loss. This property is monitored as it indicates the generalization of the model, as opposed to the loss of the training data which can potentially be indicative of the fit to the model over specifically just the training data. This callback is also particularly useful for stopping and continuing the model to periodically ensure the average NBTE Accuracy of the test dataset approximates the NBTE Accuracy of the validation dataset.

6.2.2 – Callback – Reduce Learning-Rate on Plateau + Walk-Back

The *Reduce Learning-Rate on Plateau + Walk-Back* callback was devised to combat overfitting in the trained data. While it can be acceptable for a model can to overfit in some areas if it is still improving the overall generalizable capabilities, it was observed that eventually the model would stop improving the generalizability of the model as it continued to increasingly overfit on the training partition. The response to this was to add this callback, which resets the weights to the last-best Checkpoint, with respect to the validation loss—the last best generalized model state—if no improvements were observed for a sufficiently long period of time—the patience of the callback. Additionally, to better ensure there were no skipped learning strategies, the learning rate is reduced. The patience was configured to 100 generations, where the minimum amount the validation loss is expected to improve by was $\frac{2}{l}$, where l is the average side length of the target space. This allows for larger spaces to train more slowly, given that they will have more model parameters than smaller spaces. Finally, if this Callback is triggered, then the learning rate was reduced by 10.

6.2.3 – Callback – Early Stopping

The Early Stopping callback operates similarly to the Learning-Rate Reduction callback, in which a patience is used to interrupt training if a minimum improvement of the monitored value is not met. The minimum improvement is a reduction in the loss by $\frac{2}{l}$. The patience used for the experiments is 200. This patience exceeds the patience of the Learning Rate Reduction callback, as it would be ineffective to have the Learning Rate Reduction callback be part of the model when the model would be interrupted before the learning rate could be reduced. This is used to save time, allowing another experiment to be started rather than continuing to train a model that has been unable to improve for a sufficiently long time. While it is not guaranteed that the model would not have improved if left to continue training indefinitely, the early stopping mechanism is used as an external deciding factor on when to determine it is finished.

6.3 – Initial Model Parameters

Model parameters are provided in the specifications of the best-discovered configurations for each architectural approach. The specifications for the Conv-to-ConvT, Conv-to-Dense, and Sliced Conv-to-Dense are detailed, respectively in Figure 6-1, Figure 6-2, and Figure 6-3. Parameters not specified, such as padding, are assumed to not be used in these architectural approaches. Furthermore, per Chapter 4, the activation function between each layer is given to be the standard definition of the sigmoidal activation function. Finally, model weights are randomly initialized across a uniform distribution.

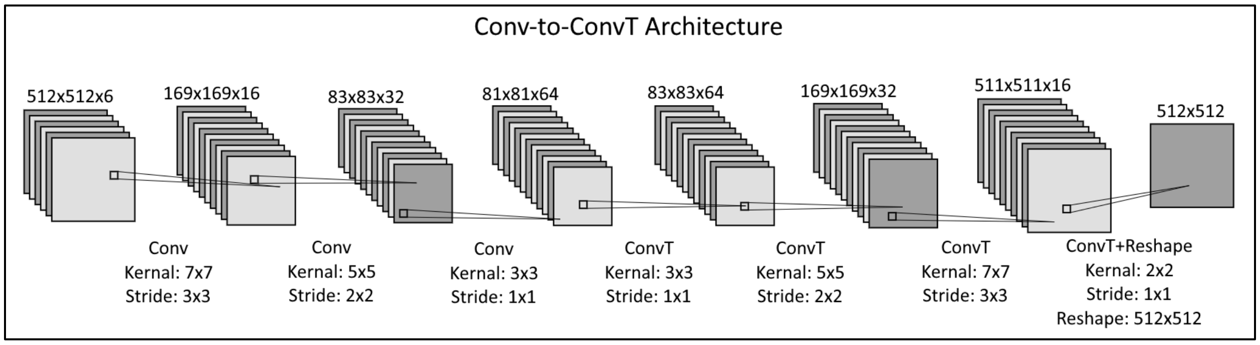
Conv-to-ConvT Architecture
512x512x6
169x169x16
83x83x32
81x81x64
83x83x64
169x169x32
511x511x16
512x512
Conv Kernal: 7x7 Stride: 3x3
Conv Kernal: 5x5 Stride: 2x2
Conv Kernal: 3x3 Stride: 1x1
ConvT Kernal: 3x3 Stride: 1x1
ConvT Kernal: 5x5 Stride: 2x2
ConvT Kernal: 7x7 Stride: 3x3
ConvT+Reshape Kernal: 2x2 Stride: 1x1 Reshape: 512x512

Figure 6-1 – Best Discovered Conv-to-ConvT Sequential Model Layout

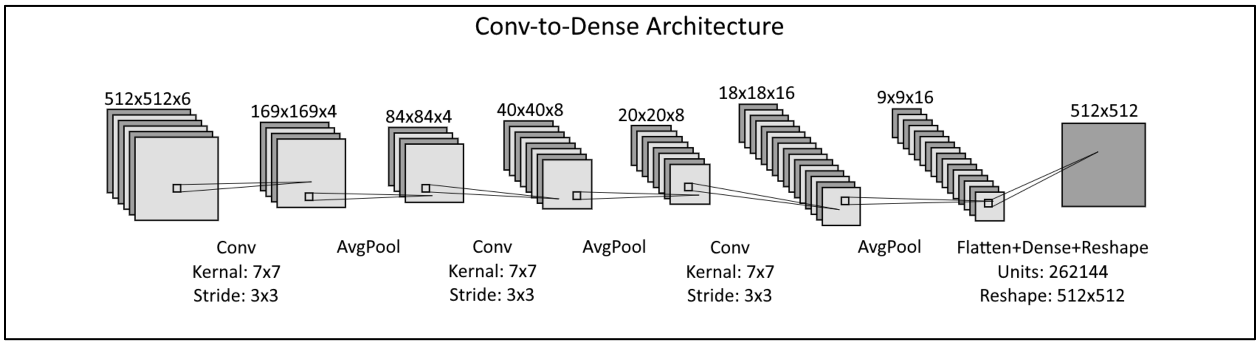
Conv-to-Dense Architecture
512x512x6
169x169x4
84x84x4
40x40x8
20x20x8
18x18x16
9x9x16
512x512
Conv Kernal: 7x7 Stride: 3x3
AvgPool
Conv Kernal: 7x7 Stride: 3x3
AvgPool
Conv Kernal: 7x7 Stride: 3x3
AvgPool
Flatten+Dense+Reshape Units: 262144 Reshape: 512x512

Figure 6-2 – Best Discovered Conv-to-Dense Sequential Model Layout

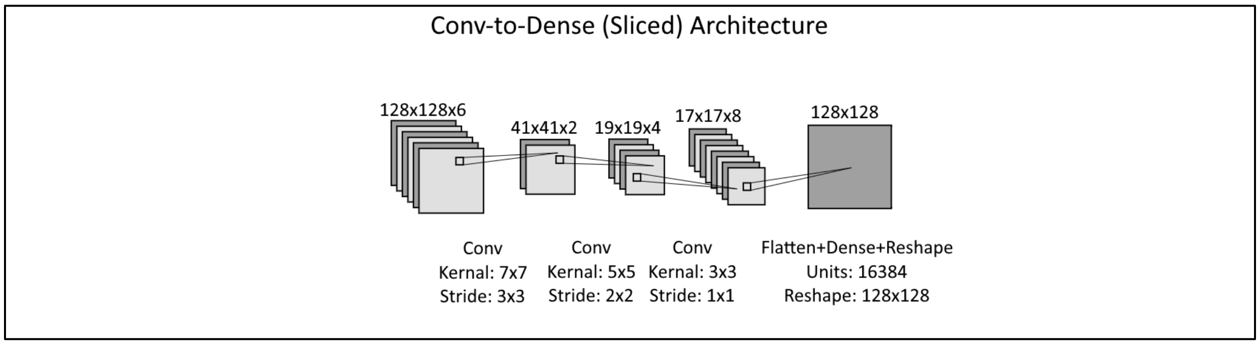
Conv-to-Dense (Sliced) Architecture
128x128x6
41x41x2
19x19x4
17x17x8
128x128
Conv Kernal: 7x7 Stride: 3x3
Conv Kernal: 5x5 Stride: 2x2
Conv Kernal: 3x3 Stride: 1x1
Flatten+Dense+Reshape Units: 16384 Reshape: 128x128

Figure 6-3 – Best Discovered Sliced Conv-to-Dense Model Architecture

Chapter 7 Experiments and Results

In the spatial models, navigational features are modeled by Convolutional Neural Networks (CNNs), which provide an internal state within a model that is passed to either Transposed-Convolutional Neural Networks (TCNNs) or Dense Neural Networks (DNNs). The TCNNs and DNNs then map the produced navigational feature maps into the navigational solution. Thus, the specifications for these architectures will entail that they intake some 3-dimensional array (HWC format) that details the presence indicators of navigational features within a region of space determined by some resolution and produce a 2-dimensional array that details the predicted navigational solution.

Input resolutions are configured to be $1 \text{ km} \times 1 \text{ km}$ regions of a projection of the navigable area onto a cartesian plane and the output navigational solution resolutions are likewise configured to be $1 \text{ km} \times 1 \text{ km}$ regions. Given the captured space is a 512^2 km^2 area and there are 6 navigational features, the Conv-to-ConvT and Conv-to-Dense architectural approaches take an input of size $512 \times 512 \times 6$ and produce an output of size 512×512. Similarly, the Sliced Conv-to-Dense approach takes in a spatially modified input and produces an equally scaled output: an input size of $128 \times 128 \times 6$ and an output size of 128×128. This exact resolution was chosen because a 4×4 of tiled captures can perfectly fit into the full captured area. Note that there are two post-processing styles for the Sliced Conv-to-Dense: 128 km offset and 32 km offset. The former is the version of which that can fit perfectly into a captured area, given the offset size is the same as the size of the output. The latter, however, leads to a type of overlapping style in which multiple overlapping predictions are made and collated into a single prediction. This latter variant tests to see if this overlapping style is an effective way to combat lossy behaviors of the 128 km

offset style, which will likely not be able to account for a proportionately higher number of features near its borders.

Because all models have a n-dimensional array input where the values of the array are constrained to the domain $\{0,1\}$ for the spatial models, the sigmoidal activation function is across every layer used due to its output range $(0,1)$ that can closely approximate this range. Note that the values are not normalized, as the vector components—channel values—of the spatial models are in the domain $\{0,1\}^6$.

These experiments empirically demonstrate the effectiveness of the proposed navigational feature modeling techniques with comparable architectural approaches that sequentially transform the modeled navigational feature data into a corresponding navigational solution. Using the datasets produced by the airspace scenario generator and the actual navigational solutions that are deterministically derived from the simulated data, the effectiveness is then derived by comparing the predictions of the fitted models against the actuals. Note that the architectural implementations demonstrated detail the best discovered variants of those style of approaches with regards to these experiments; they demonstrate the core concept of the proposed architectures by exemplifying an effective approach for a specific problem domain.

These architectures are evaluated with a series of statistical metrics and analytical techniques and are detailed along with the exact architectural specifications of the models selected to demonstrate the learnt capabilities. Confidence in the effectiveness of the approach will then be evaluated by a number of factors, including:

1. NBTE Accuracy and Coherence scores. These are further compared to Cosine Similarity Accuracies to provide a traditional basis of evaluation and comparable accuracy metric.

$$\text{cosine similarity accuracy} = -\text{cosine similarity loss}$$

Where the accuracy is then the displacement of the loss from the center point of the interval $[-1,1]$. Accuracies $> 0\%$ then indicate some positive similarity and $< 0\%$ some opposite.

2. Similarity of sampled avoidance means and standard deviations to identify similarities in the distributions of the actual and predicted navigational solutions.

$$\text{similarity} = 1 - \frac{|x_0 - x_1|}{x_0 + x_1} \tag{7-1}$$

Where x_0, x_1 then either detail the aggregated avoidance mean or standard deviations for both the actual and predicted, respectively.

Additionally, select filters of the CNN layers, used to model the navigational features, will be presented to provide interpretations on strategies and compare qualitative similarities between architectures to make some initial hypotheses on their formation.

7.1 – Baseline Scores

Baseline scores are empirically observed from the testing partition of the dataset. Given the sufficiently high number of data samples, the distribution of both the sample means and standard deviations approximate the normal distribution. Given 200 scenarios, this then provides more than adequate data sampling, per the Central Limit Theorem. To further empirically evaluate the Neutral Accuracy scores, to calculate the NBTE Coherence scores, each scenario is sampled with 30 tests of random prediction realizations in the distribution in $\mathbb{U}(0,1)^{512\times512}$.

Finally, note that sliced architectural models are evaluated with respect to their tiled prediction against the entire space, not just a slice of a solution, as they are meant to be a constructive tool.

As such, no Neutral Accuracy needs to be determined with respect to the expected value of a randomized prediction in the distribution $\mathbb{U}(0,1)^{128\times128}$.

Thus, the following values are empirically observed:

$$\text{neutral loss} \cong 0.68891 \tag{7.1-1}$$

$$\text{neutral accuracy} \cong 0.31109 \tag{7.1-2}$$

$$\bar{\mu}_0 \cong 0.18184 \tag{7.1-3}$$

$$\bar{\sigma}_0 \cong 0.23269 \tag{7.1-4}$$

Where $\bar{\mu}_0$ and $\bar{\sigma}_0$ are the averaged means and standard deviations of the scenarios in the testing partition of the dataset. These values are used to evaluate the similarities in those properties of the distribution of avoidances between the predictions of the trained models and actuals.

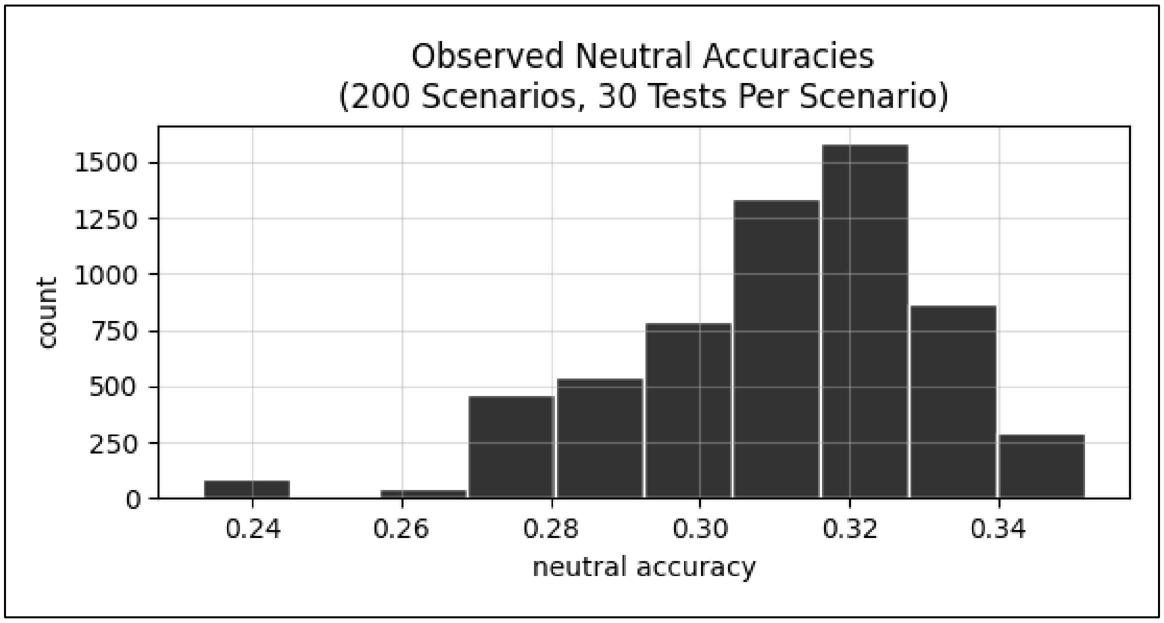

Figure 7-1 – Empirically Observed Neutral Loss Histogram (Test Dataset)

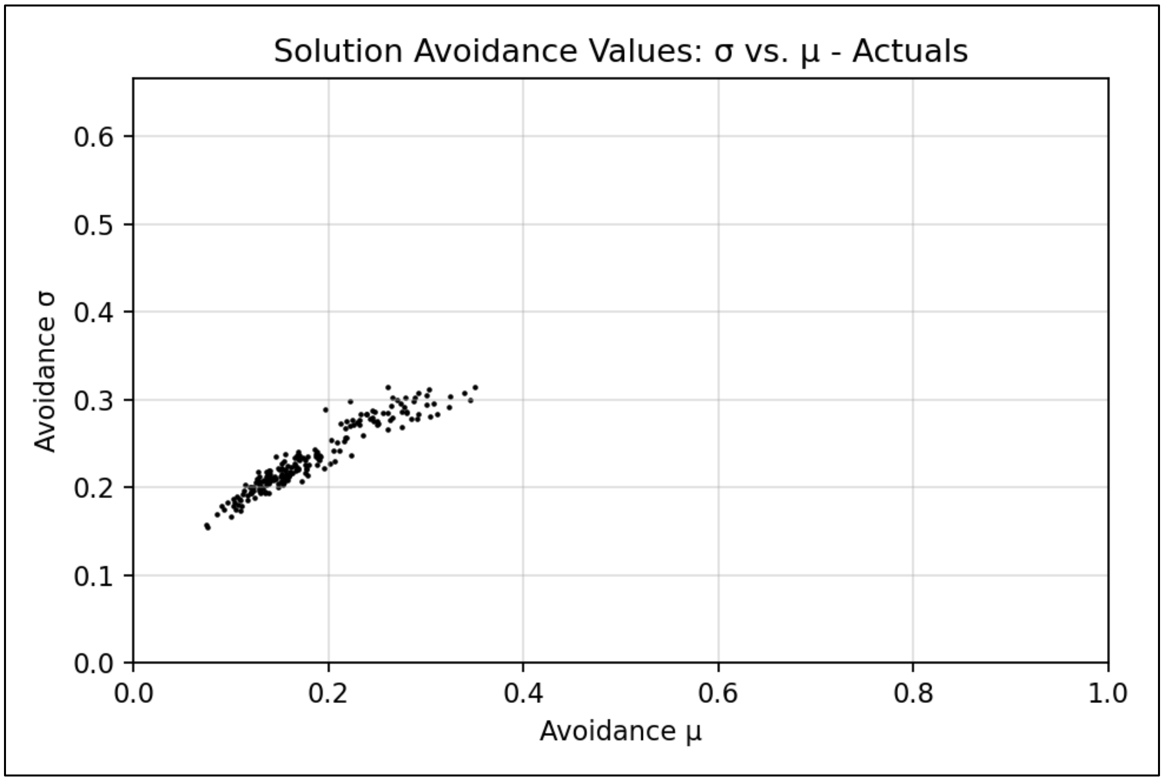

Figure 7-2 – Test Scenarios Avoidance σ vs. μ

7.2 – Quantitative Summaries

This section is a foreword of quantitative observations made for reference throughout the remainder of the experiments. It tabulates the NBTE and Cosine Similarity scores, as well as details the distribution properties similarities for reference. It will also compare the NBTE scores with respect to the Cosine Similarity scores, to identify how they differ.

Architecture		NBTE Loss	NBTE Accuracies			NBTE Coherence
			Avg	Min	Max	
Conv-to-ConvT		29.573%	70.427%	57.838%	78.379%	57.073%
Conv-to-Dense		30.313%	69.687%	58.700%	78.900%	55.999%
Sliced	128km Offsets	37.736%	62.264%	50.917%	72.626%	45.224%
Conv-to-Dense	32km Offsets	39.326%	60.674%	47.136%	67.872%	42.915%

Table 7-1 – Model NBTE Scores

Architecture		Cosine Similarity Accuracies		
		Avg	Min	Max
Conv-to-ConvT		80.581%	63.102%	93.359%
Conv-to-Dense		78.841%	57.907%	92.911%
Sliced	128km Offsets	74.520%	56.539%	87.271%
Conv-to-Dense	32km Offsets	72.609%	52.745%	85.580%

Table 7-2 – Model Cosine Similarity Scores

Architecture		μ		σ	
		avg	similarity	avg	similarity
Conv-to-ConvT		0.27193	80.146%	0.20143	92.799%
Conv-to-Dense		0.27749	79.176%	0.24399	97.629%
Sliced	128km Offsets	0.30348	74.937%	0.18980	89.848%
Conv-to-Dense	32km Offsets	0.33911	69.811%	0.18040	87.340%

Table 7-3 – Prediction Avoidance Means (μ), Standard Deviations (σ), and Similarity to Actuals

Table 7-1 and Table 7-2 detail the NBTE and Cosine Similarity Accuracies, respectively. In general, the two detailed accuracy metrics are proportional, with one exception. The max accuracy scores between the Conv-to-ConvT and Conv-to-Dense show a discrepancy: the max NBTE Accuracy score for the Conv-to-ConvT architecture is less than the max score for the Conv-to-Dense architecture, however the Cosine Similarity Accuracies for the same metrics do not share this inequality. This is a seemingly negligible difference as the two scores are very similar, but it demonstrates the potential for differing losses and accuracies. Furthermore, while the metrics are otherwise proportional, the variance of the two measures is notably different, with the Cosine Similarity Accuracy also having a wider range of maxes and mins. Using these two metrics, it is also possible to infer there is substantial bias in the predictions still; the Cosine Similarity Accuracies are notably higher than the NBTE Accuracies in each category, thus indicating that the predictions are not properly matching the distribution of avoidance values. If there was less bias in the predictions, the two accuracies would be fairly comparable where any error would then be balanced deviations within the distribution.

The variation in distributions is further identified in Table 7-3; the Conv-to-Dense architecture details an average avoidance-value mean similar to the other architectures, but notably has an accurate standard deviation. While it would appear that a linear translation could better improve the results, this attempted post-processing step actually yielded more error than without. This indicates there are additional distribution parameters that cannot be easily accounted for, likely representing the complex behaviors when designing the avoidance gradients to prevent such simple linear optimization strategies from being the answer. Offsetting the avoidance values in the Conv-to-Dense architecture's predictions, with respect to the difference between the evaluated mean of the avoidance values of the Conv-to-Dense architecture's predictions and the empirically determined mean of the distribution, yields the NBTE scores in Table 7-4.

Architecture	NBTE Loss	NBTE Accuracies			NBTE Coherence
		Avg	Min	Max	
Conv-to-Dense (Modified)	32.810%	67.190%	59.411%	73.115%	52.374%

Table 7-4 – NBTE Scores w/ Modified Predictions

7.3 – Conv-to-ConvT

Sampled Conv-to-ConvT predictions are detailed in Figure 7-3, with a historic loss plot in Figure 7-4, and empirical avoidance distribution mean and standard deviation in Figure 7-5.

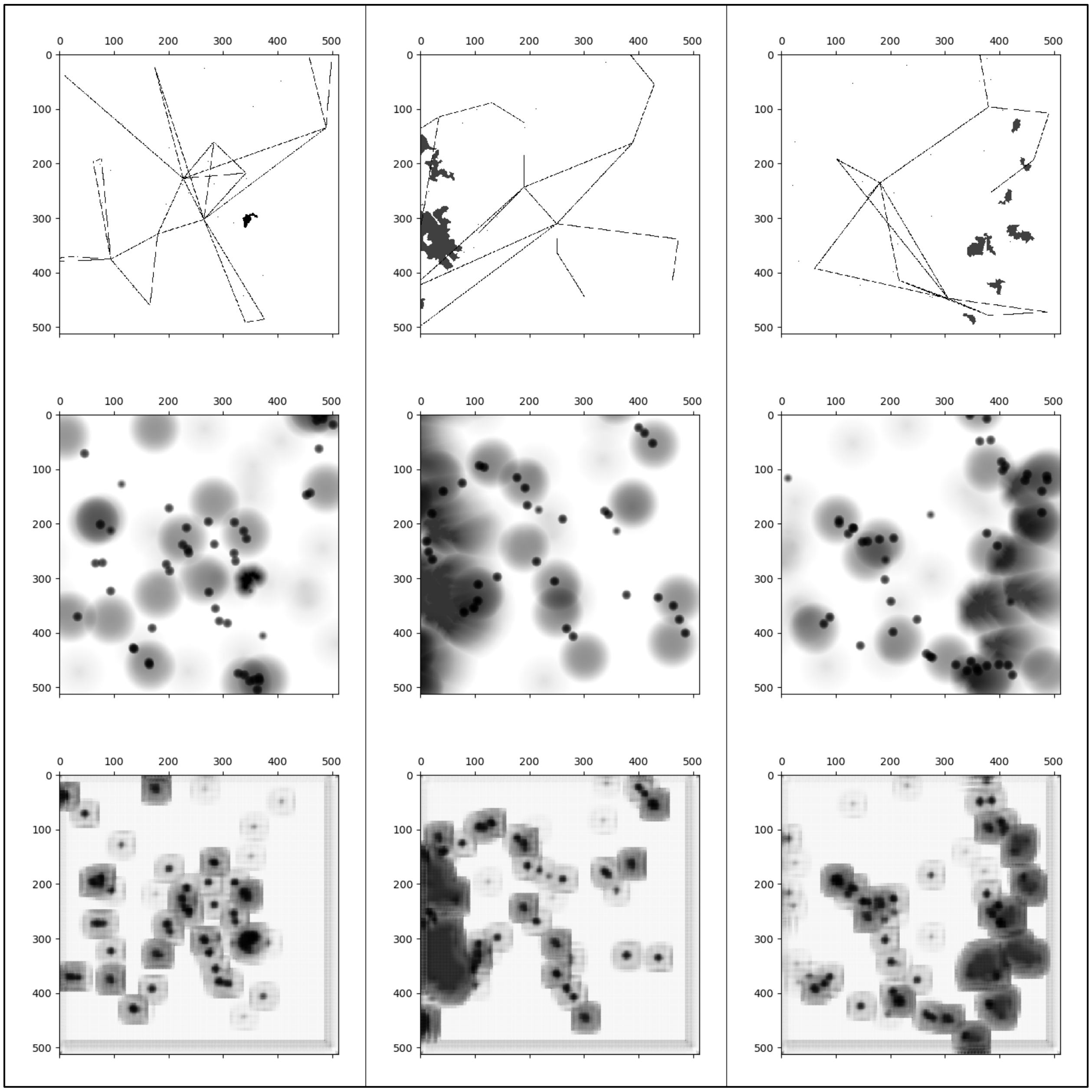

Figure 7-3 – Conv-to-ConvT: Scenario (Top), Actual (Middle), and Predicted (Bottom)

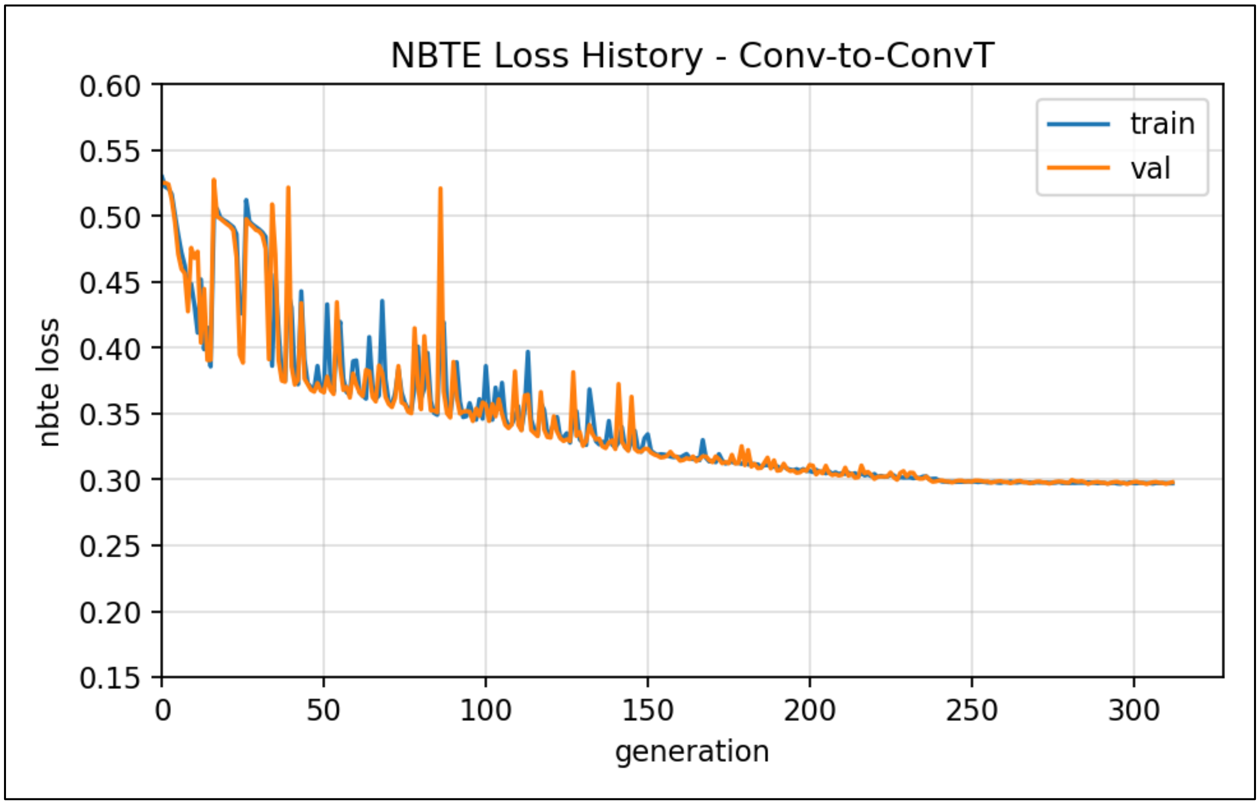

Figure 7-4 – Conv-to-ConvT NBTE Loss History

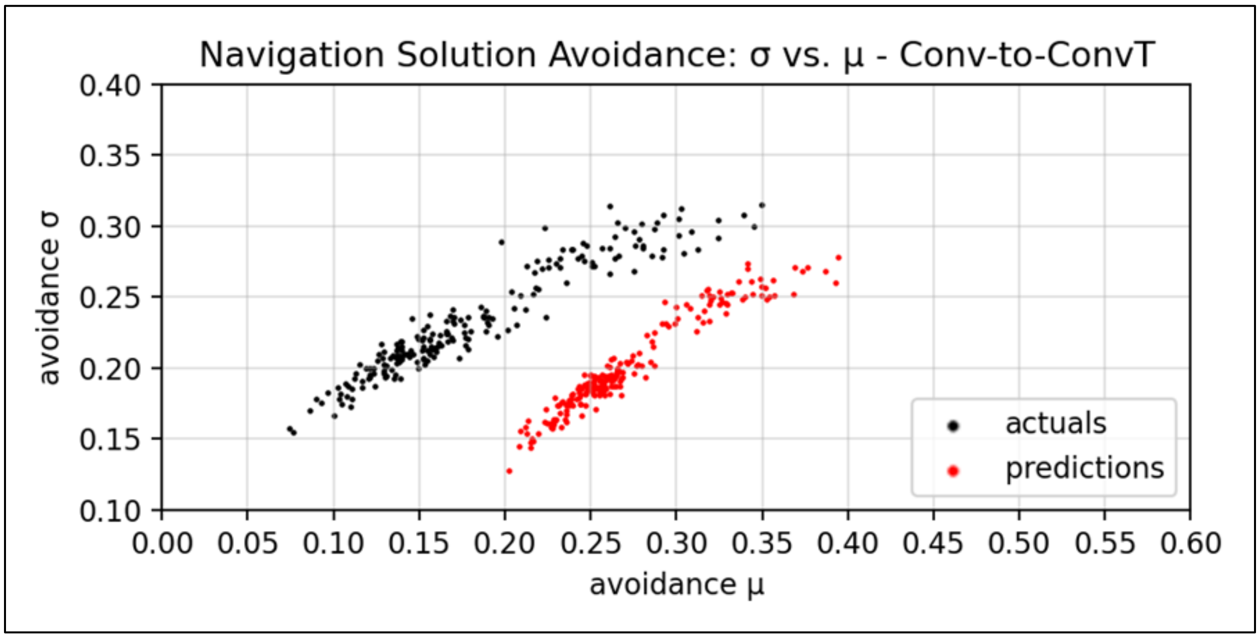

Figure 7-5 – Conv-to-ConvT Prediction Avoidance μ vs. σ

The Conv-to-ConvT architecture scored an average NBTE Accuracy of 70.427%, which informs of a NBTE Coherence of 57.077%. Historic NBTE Loss data details practically no overfitting of the data, where both then the training and validation scores show nearly identical losses with each post-generation evaluation of the model. Model predictions detail amoeba-like approximations of the avoidance gradients that somewhat represent the actual distributions. The similarity in the aggregated avoidance values detail a mean similarity of 80.146% and standard deviation similarity of 92.799%. However, as evidenced by Table 7-4, there are additional nonlinear distribution parameters beyond the distribution mean and standard deviation.

Qualitatively analyzing the model predictions, they are capable of producing avoidance gradients for nearly all captured navigational features and detail border-artifacts to combat localized influence from uncaptured navigational features. The avoidance gradients are, however, fairly simplistic; they detail two near-constant radial avoidances that emulate the steps of avoidance gradients with diminishing strengths. The radius of the inner circle is then capable of approximating some parameters of the falloff function when compared to the radius of the outer circle. However, this is still a local loss minimum. Border-artifacts detail a constant border on the lower and righthand sides of the predictions, instead of a uniform approach, which is presumed to indicate some stochastic bias in the scenarios where those sides correlated to the greatest amount of unknown border noise.

A filter used to build the initial feature maps that directly model the navigational features in the demonstrated Conv-to-ConvT architectural approach are presented in Figure 7-6, where then each input channel corresponds 1:1 with a navigational feature. Some pattern can be observed in these initial filters. Note that not all convolution filters are displayed due to either large filter counts, they are sufficiently similar to other presented filters, or they largely comprise of sufficiently complex design that they appear to be random noise.

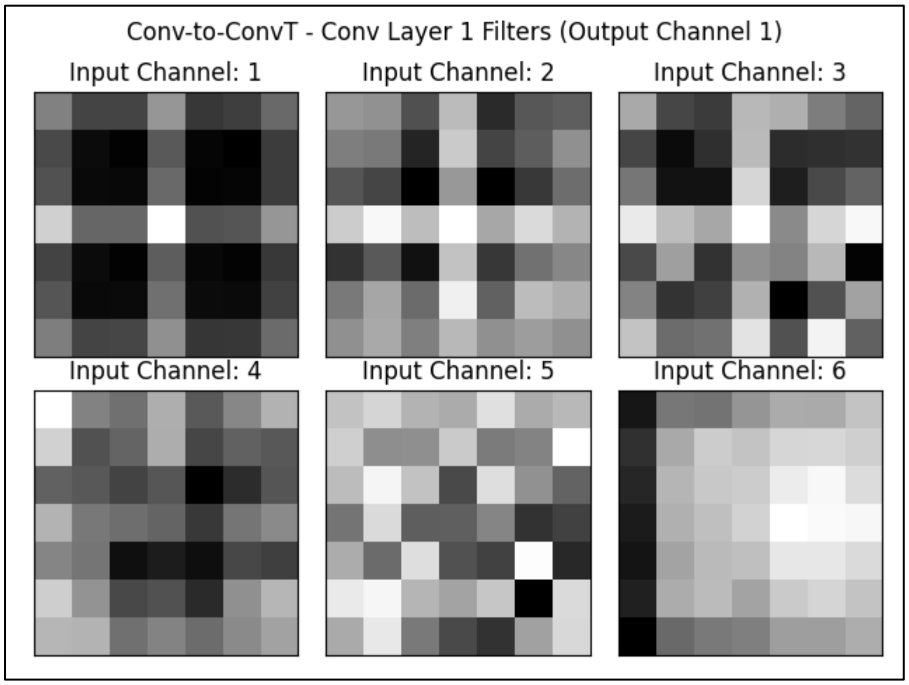

Figure 7-6 – Conv-to-ConvT Conv Layer 1 Filters (to Output Channel 1)

Recall the channel ordering of presence indicators mentioned in §3.3 – Discretizing Process, which was given as: 1) Airfields, 2) Aircraft, 3) Miscellaneous Type A Hazards, 4) Miscellaneous Type B Hazards, 5) No-Fly Zones, and 6) Weather Fronts. The filters for the input channels 1 and 6 are observed to undergo minor variations for all the output layers, detailing a fairly stable recognition algorithm. This is especially interesting for channel 1 as it corresponds to a point-based feature that shares the same shape, albeit a different count-distribution, as the other point-based features, yet is not represented in any of the other point-based filters even along different output channels. Filters for the input channels 2-5 have some structure, where filters along channels 2 and 3 bear some vague similarity to the channel 1 filter, but appear to be fairly noisy.

Filters for the remainder output channels, with respect to the input channels 2-5, detail similar structures, but significantly more variance than the filters for the input channels 1 and 6 experience. Asides from the filter for input channel 6, it is unclear how exactly these filters are optimized. Whereas for channel 6 the vertical bar indicates some capability to map the outline of a discovered weather front. Especially given by how this filter is relatively unchanging across multiple output layers, this provided a consistent viable strategy. The remaining CNN filters in layers 2 and 3 detail seemingly noisy filters that are fitted to map some relation between the output of this detailed first layer output and the corresponding navigational solution.

Some example outputs of the demonstrated Conv-to-ConvT architecture are detailed in Figure 7-3. These sampled predictions detail amoeba-like stylizations of the avoidance gradients. It is hypothesized that the transposed-convolutions were not able to perfectly learn the mappings from the navigational feature map produced by the convolutions to the navigational solutions. As a result, they identified a local minimum gradient that allowed a balanced representation of the navigational solutions. However, to their credit, they are able to identify multiple parts of these navigational solutions; note the darker inner regions of the captured navigational hazards; there is a capability to account for some points along the gradient. Additionally, this architectural approach effective captures most navigational solutions, where the missed ones appear typically appear in proximity to other features. However, it is not able to capture directional gradients, such as demonstrated by the weather-front in the example on the right. Recall earlier mentions about the captured area of the navigational area being surrounded by an uncaptured region, which is still populated with features. The avoidance gradients of these features add unobserved noise to the border of the captured region, which is then hypothesized as to why the bottom and righthand sides of the predictions are occupied by that constant artefact border; the artefact is another local minimization strategy that attempts to reduce the loss incurred from unobserved features on the other side of the border.

7.4 – Conv-to-Dense

Sampled Conv-to-Dense predictions are detailed in Figure 7-7, with a historic loss plot in Figure 7-8 and empirical distribution mean and standard deviation in Figure 7-9.

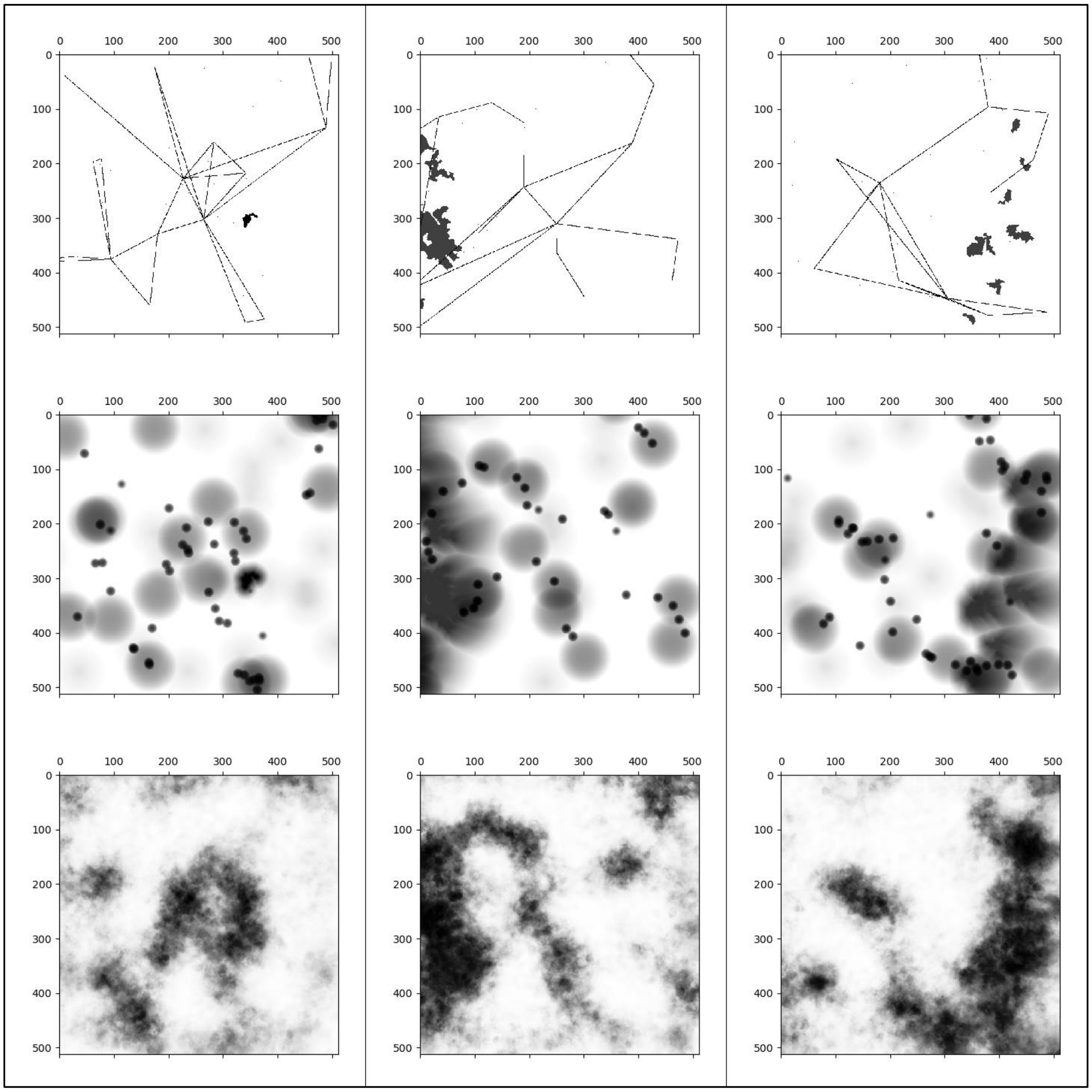

Figure 7-7 – Conv-to-Dense: Scenario (Top), Actual (Middle), and Predicted (Bottom)

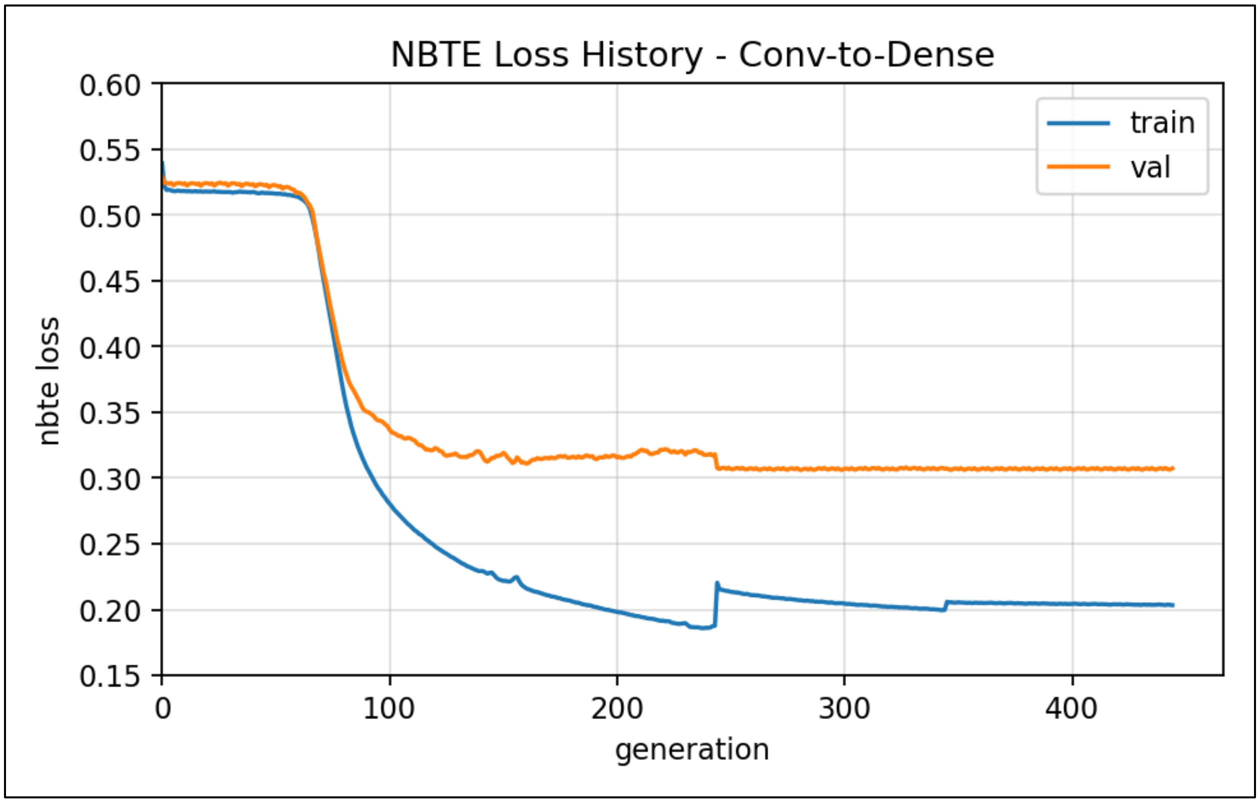

Figure 7-8 – Conv-to-Dense NBTE Loss History

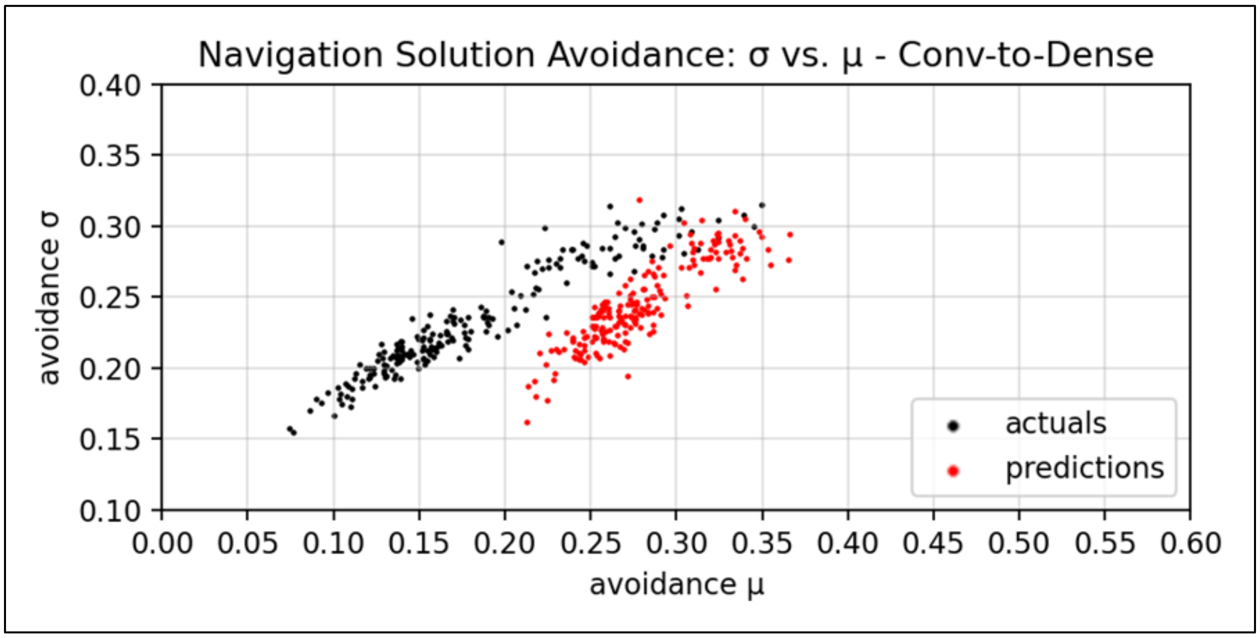

Figure 7-9 – Conv-to-Dense Prediction Avoidance μ vs. σ

The Conv-to-Dense architecture scored an average NBTE Accuracy of 69.687%, which informs of a NBTE Coherence of 55.999%. Historic NBTE Loss data details significant overfitting of the data, which highlights the attempt of the Reduce LR + Walk-Back Callback at around generation 250 and again at 350, which details some success in further improving the validation loss—the generalizability of the model. Model predictions detail fuzzy approximations of the avoidance gradients that somewhat represent the actual distributions. The similarity in the aggregated avoidance values detail a mean similarity of 79.176% and standard deviation similarity of 97.629%. However, as evidenced by Table 7-4, there are additional nonlinear distribution parameters beyond the distribution mean and standard deviation.

Qualitatively analyzing the model predictions, they are capable of producing avoidance gradients for most larger clusters of avoidances and of the polygonal features with substantially larger areas. There are some border-artifacts to combat localized influence from uncaptured navigational features, however not as heavy as the Conv-to-ConvT approach and the artifacts here are more uniformly applied to each border. The quality of the border gradients is also improved over Conv-to-ConvT in places, and is notably able to more accurately model the complex, non-uniformly radial shapes of the weather fronts.

Some filters used to build the initial feature maps that directly model the navigational features in the demonstrated Conv-to-Dense architectural approach are presented in Figure 7-10. These two filters are selected because, out of the 4 output channels the initial CNN layer produces, all point-based filters—input channels 1-4—detail minor variations of these two patterns, indicating a clear capability to model point-based features effectively. This coincides with the results of the Conv-to-ConvT observations where even though the modeled feature is a dot in the space, with respect to the channel, there is some underlying optimization that allows for these varying filters to better represent the information.

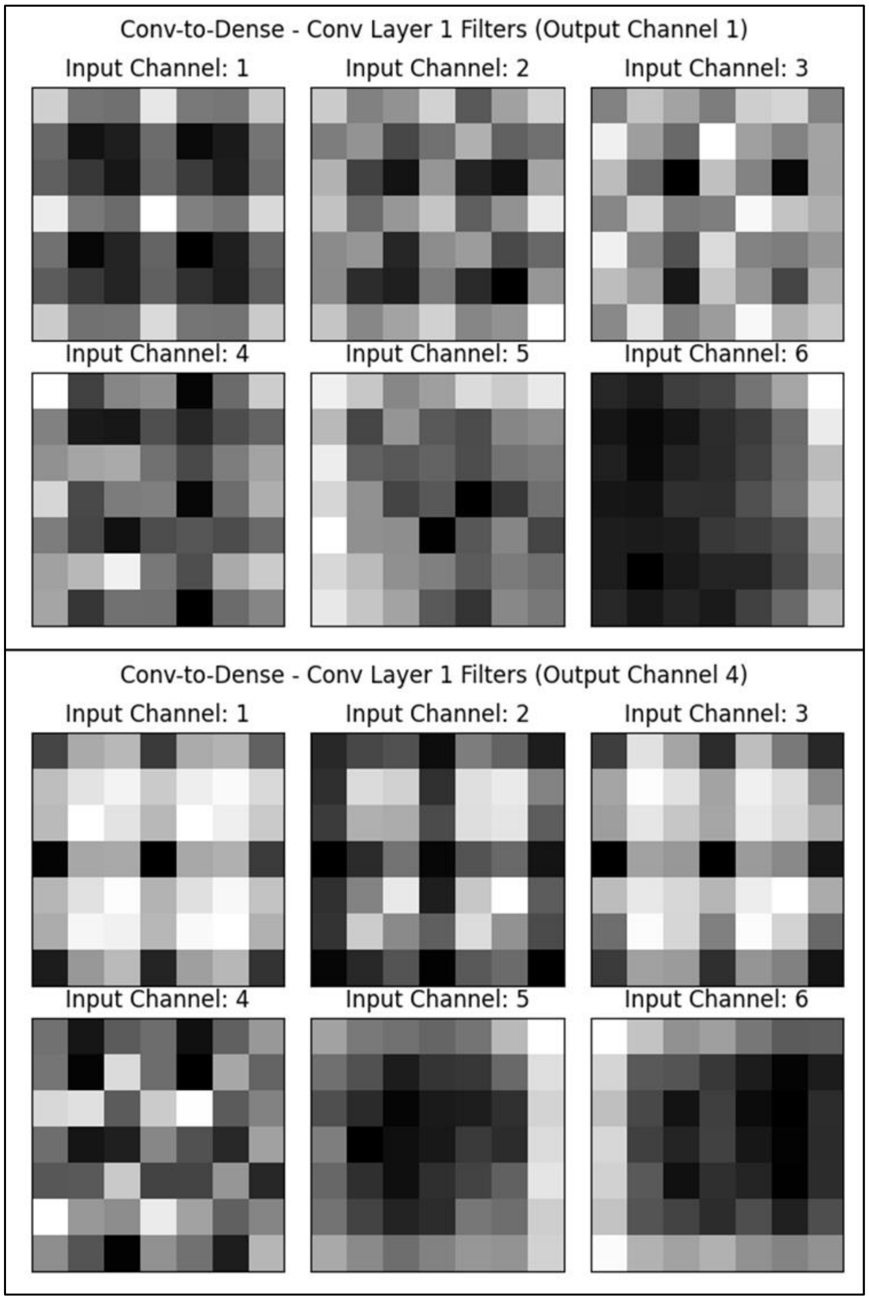

Figure 7-10 – Conv-to-Dense Conv Layer 1 Filters (to Output Channel 1 and 4)

The differences in the filters, compared to the Conv-to-ConvT model, is likely related to how the features are modeled with respect to how the internal modeled states are transformed into the navigational solution; they are dependent on the corresponding avoidance gradient of each navigational feature. For the polygonal features, while the gradients slightly differ note that both output channel 4 filters are a rotation of the output channel 1 filters. Similarly, channels 2 and 3 detail other rotational variants. This is likely then to help identify the edges of the polygonal features with respect to the avoidance gradients of their respective navigational features.

7.5 – Sliced Conv-to-Dense

Sampled Conv-to-Dense predictions are detailed in Figure 7-11 and Figure 7-12, with a historic loss plot in Figure 7-13 and empirical distribution mean and standard deviation in Figure 7-14.

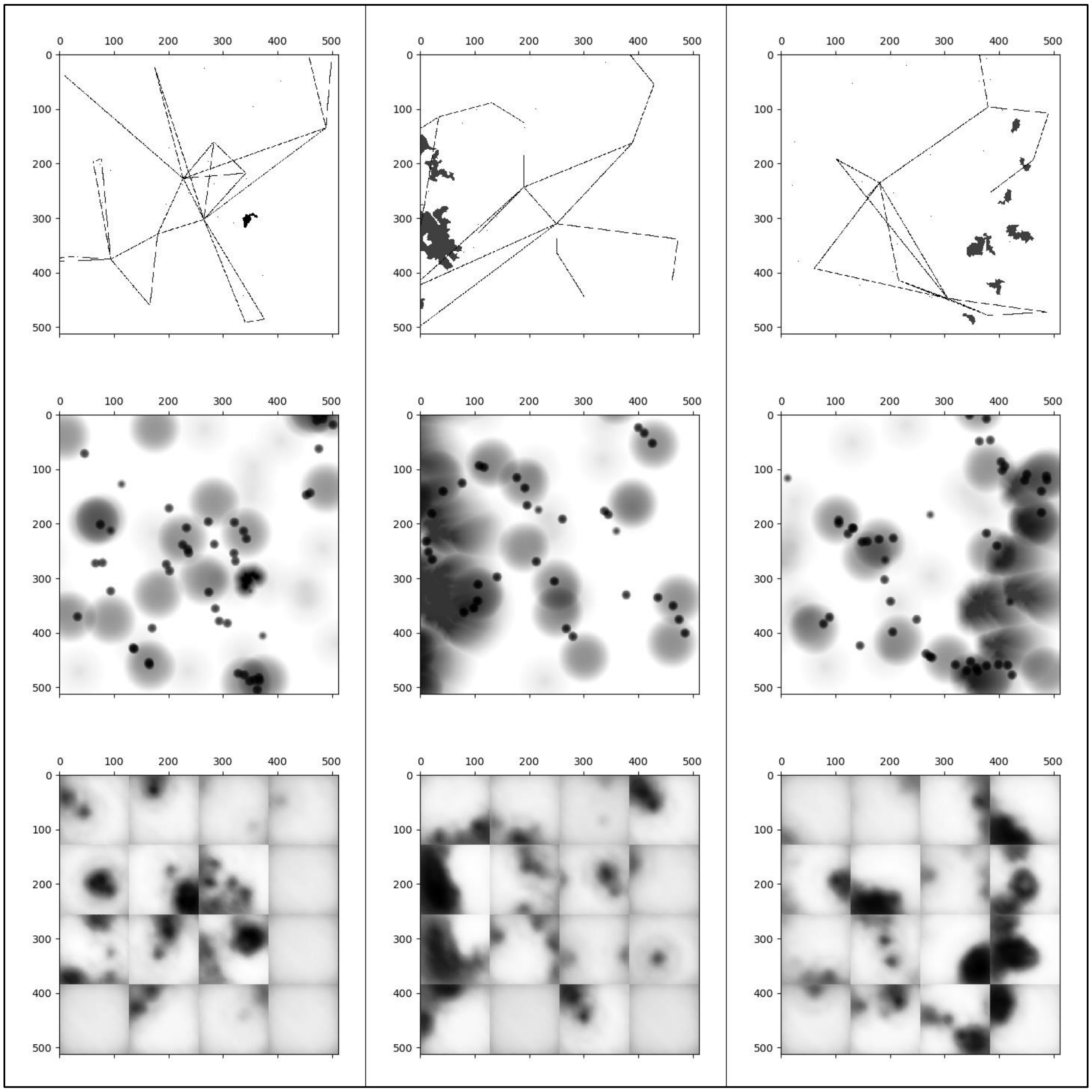

Figure 7-11 – Sliced Conv-to-Dense (128km Offsets): Scenario (Top), Actual (Middle), and Predicted (Bottom)

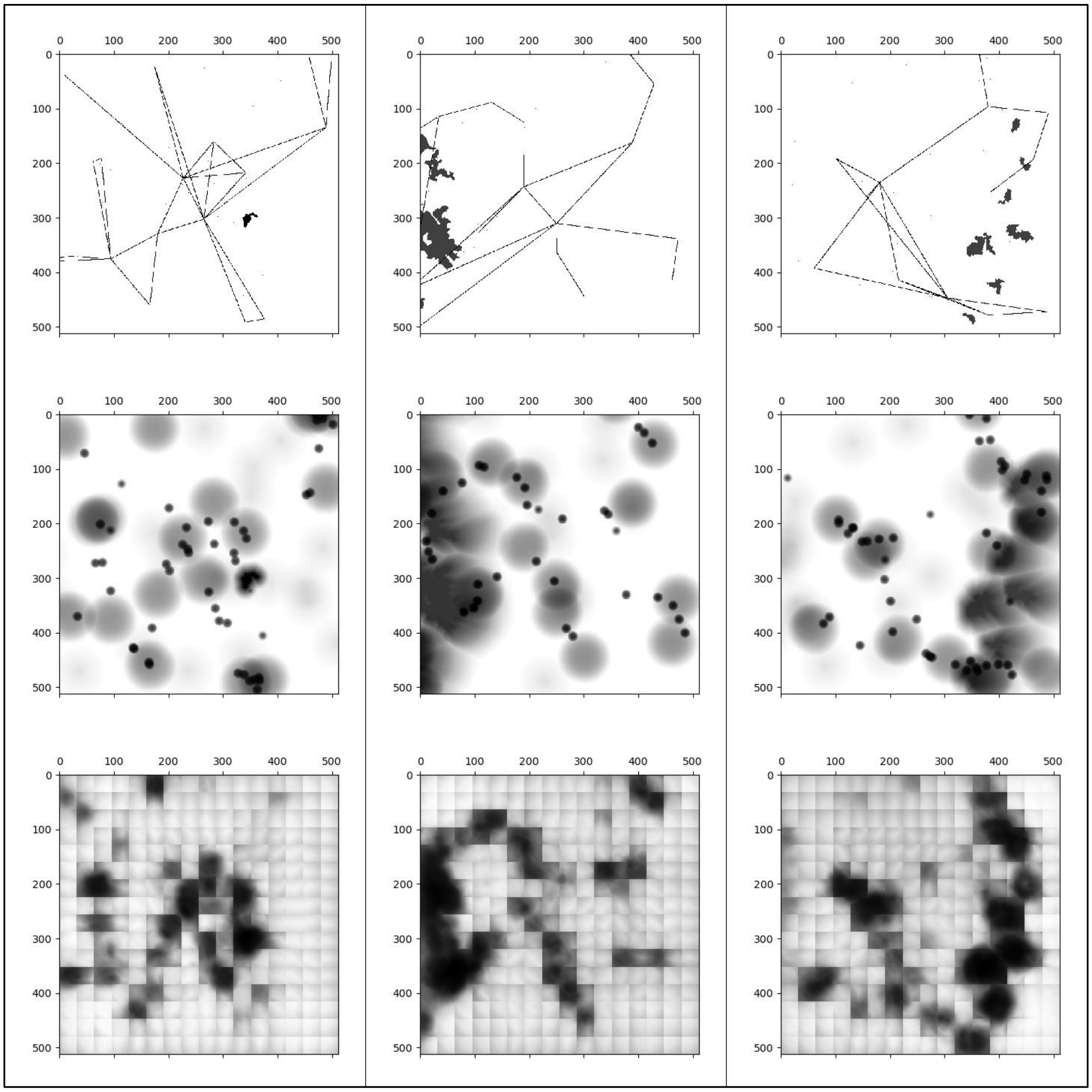

Figure 7-12 – Sliced Conv-to-Dense Model (32km Offsets): Scenario (Top), Actuals (Middle), and Predicted (Bottom)

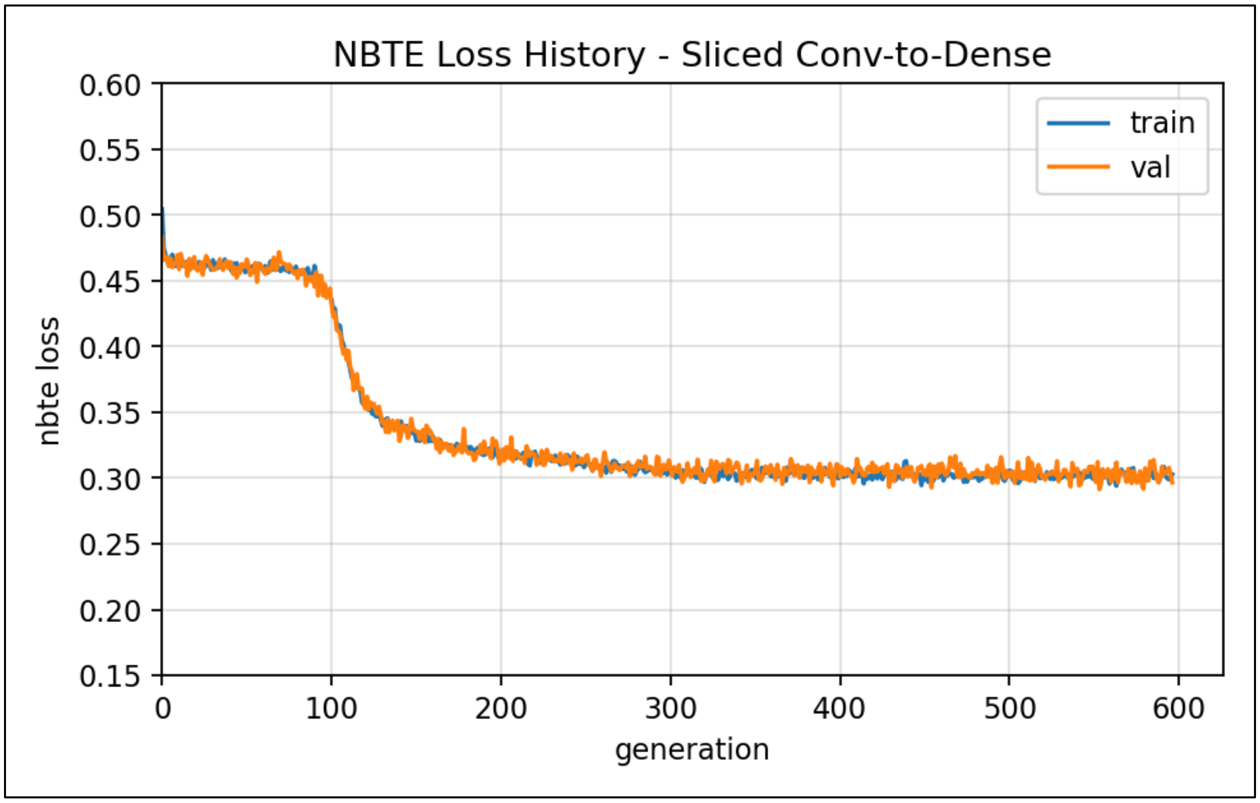

Figure 7-13 – Sliced Conv-to-Dense NBTE Loss History

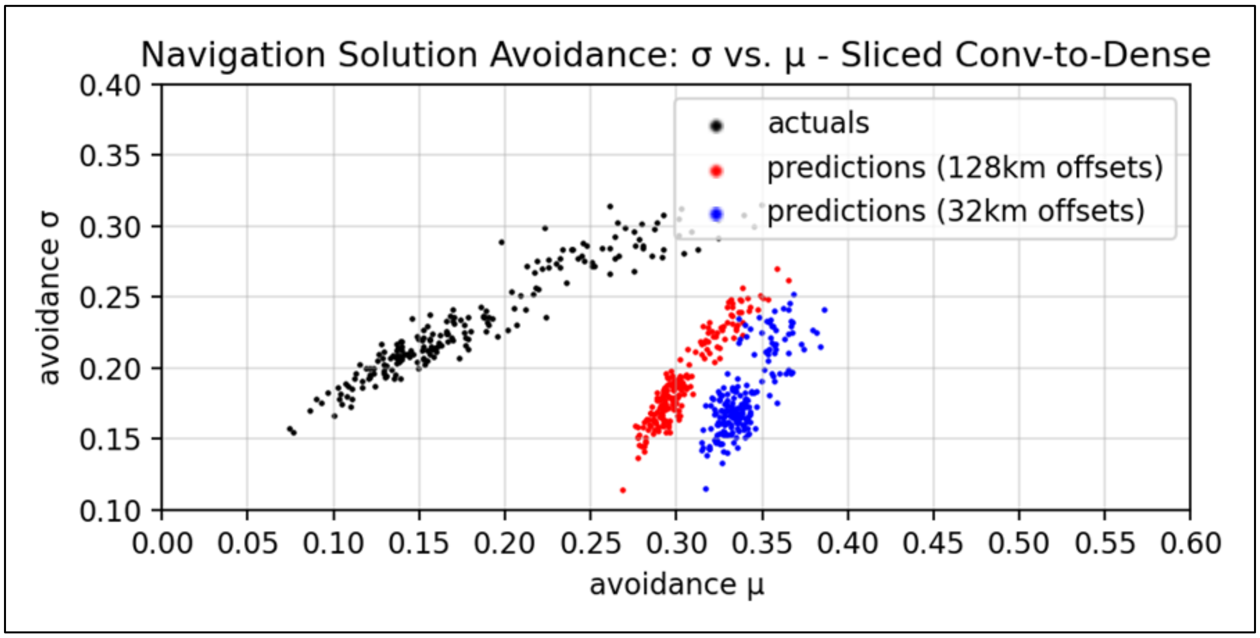

Figure 7-14 – Sliced Conv-to-Dense Prediction Avoidance μ vs. σ

The Sliced Conv-to-Dense architectural approach details a singular architecture with multiple post-processing strategies: tiling offsets.

1. A 128km offset strategy, which results in a minimum count of navigational solutions being generated as the resulting $128\text{km} \times 128\text{km}$ sized predictions can be fit into a 4×4 arrangement to recreate the target $512\text{km} \times 512\text{km}$ size of the navigational solution.
2. A 32km offset strategy, where each tile will have overlap with multiple tiles and the highest avoidance values of those overlapping regions are then selected to better account for neighboring navigational features. Note that this yields a 13×13 predictions to fill the 512×512 target space.

Though both approaches are derived from the same architecture, they denote two separate accuracies given these two different post-processing approaches. The 128km and 32km offset strategies detail respective average NBTE Accuracies of 62.264% and 60.674%. These correlate to the respective NBTE Coherences of 45.224% and 42.915%. While these accuracies are fairly similar, there is a clear advantage for the approach with exact tiling—the 128km offset approach. Historic NBTE Loss data details, for the architecture, a similar pattern of learning as the Conv-to-Dense approach. However, there is no overfitting here. This difference is conjectured to be the result of reducing model parameters, which reduced the capability of the model to store information to overfit on certain problems.

Qualitatively analyzing the model predictions, they are a lesser variant of the Conv-to-Dense; they have a similar style, but lack the accuracy. While they are capable of producing avoidance gradients for most larger clusters of avoidances and of the polygonal features with substantially larger areas, the cutoffs when a new tile starts effectively nullify this capability.

Where then the tiling with overlap strategy was supposed to help, the tiles simply maximized the border artifact noise that instead reduced overall performance. There are exceptions to this, such as the far righthand side prediction, where the area to the right of the weather front is fairly clean, but this is simply an impact of the tile overlapping not as heavily impacting areas near the edges.

Some filters used to build the initial feature maps that directly model the navigational features in the demonstrated Sliced Conv-to-Dense architectural approach are presented in Figure 7-15. Note the similarity of these filters to those in the base Conv-to-Dense architecture. In contrast, even though this architecture is trained with fewer filters there is no alternative patterns; the other filter for this first CNN layer details the same filters for each input channel, with minor variations. However, given the added lossy results of the model, this is then presumed to be the cause, as it informs the final predictions that are used to evaluate the accuracies.

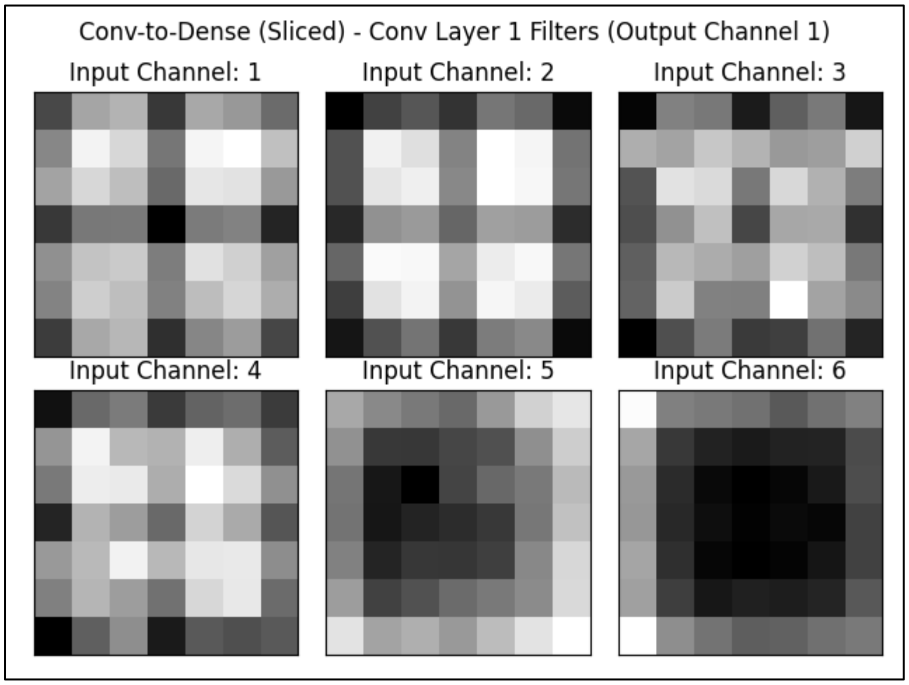

Figure 7-15 – Sliced Conv-to-Dense Conv Layer 1 Filters (to Output Channel 1)

The two figures, Figure 7-11 and Figure 7-12, detail the predictions from the Sliced Conv-to-Dense architectural approaches with the two tiling offset strategies: 128km and 32km, respectively. These figures highlight the main source of loss: border artefacts. Each border is clearly distinct, due to the high levels of baseline fill used to minimize the border avoidance during training, with the exception of the righthand side of the prediction in the right column. Especially with the overlapping tiling approach, the borders are fairly seamless and do well with regards to not utilizing border noise and still capturing most of the features. However, as the resolution of the tiling strategy is adjusted such that more tiles are needed, the computational complexity begins to make this more infeasible as it can be computationally expensive to generate all the tiles required.

Chapter 8 Temporal Navigational Features and Solutions

This chapter serves as a basis of future efforts to extend the application of the developed models to include temporal navigational feature modeling. Temporality can be an important step forward in creating more efficient pathing determinations, which need fewer corrective actions due to earlier determinations about future conditions. This section details some experiments in exploring this domain, which utilizes the proposed navigational feature modeling methodologies to do so. In contrast to the spatial models that transform modeled navigational feature maps into navigational solutions, the explored temporal architectural models attempt to model spatial-temporal navigational feature maps and transform them into forecasted spatial feature maps. Subsequently, such forecasted navigational feature maps could yield navigational solutions. Initial experiments are unsuccessful, where the models learnt the weighted paths of traveling aircraft rather than how to forecast future positions. Further domain exploration is then required, as it is likely not as simple as extending the spatial architectural methodologies. However, in the meanwhile the output of the explored temporal architecture can detail partial-temporality, which can be applied to spatial scenario data to account for, to some extent, generalized future behaviors.

8.1 – Temporalized Behaviors

The temporalization of airspace scenarios is a process that utilizes the information generated in the procedural generation of airspace as an initial timestep, and then applies motion models and other temporal dynamics to the captured navigational features. Currently, this is limited exclusively to aircraft motion models, which detail simple deterministic navigation systems and stochastic decision-making systems. The result is an airspace full of autonomous aircraft that perpetually navigate the space, where they initially continue along the flightpath they were distributed along when first instantiated. Once they reach their destination airfield, a random airfield directly reachable from their current is selected and, with some controlled noise in their motion models, they continue onwards to their next destination.

8.2 – Aircraft Motion Model

When an aircraft starts a new flightpath, it determines a random unit vector u_{noise} that is directionally scaled by a variable noise level. This noise is added to the position of the destination of the aircraft, and it is that point the aircraft directly heads towards. The maximum noise potential reduces as the aircraft approaches its destination, resulting in an arcing parabolic trajectory. This trajectory helps spread out air traffic along the same flightpath and simulates how aircraft do not necessarily head straight from point-to-point.

When an aircraft starts a new flightpath, it determines a random speed s (units in the form distance per interval between timesteps) that details for each timestep how far the aircraft advances towards that offset point.

$$s \sim U(720, 1200)\ \text{km/h} \tag{8.2-1}$$

Given an originating airfield position i, destination airfield position j, random destination offset unit vector u_{noise}, speed s, and aircraft position k_t at timestep t. The targeted position relative to the destination is then denoted as v_{noise} and is a function of $i, j, k_t, u_{\text{noise}}$ and is detailed in Eq. (8.2-2).

$$v_{\text{noise}} = u_{\text{noise}} \sin\left(\pi \frac{|j - k_t|}{2|j - i|}\right) \min\left(\frac{|j - k_t|}{2}, |j - i|^{0.6}\right) \tag{8.2-2}$$

Where the direction of advancement for the next time step is denoted by u and is a function of j, k_t, v_{noise} and is detailed in Eq. (3.9.1-3).

$$u = j + v_{\text{noise}} - k_t \tag{8.2-3}$$

Finally, the position of aircraft in the next timestep is then denoted by k_{t+1} and is simply the current position offset by the direction of advancement by the speed of the aircraft.

$$k_{t+1} = su + k \tag{8.2-4}$$

If the aircraft is close enough to its destination that with speed s it could reach it by traveling in a straight line, that aircraft switches to a new flightpath originating from the previous destination. For the first step on this new flightpath, the speed s is reduced by the amount that it would have taken to reach the airfield. Speed can be reduced to a minimum of 0, so the aircraft does not fly backwards. That reduced speed is temporary, and is restored after first timestep of a new path.

Note that the specifications of the navigable spaces for the airspace scenarios still apply here. Therefore, it is possible for airfields to be placed outside of a region of captured space, where subsequently they then must be connected to an airfield inside the captured space, as there is an extremely high probability there is at least one. Furthermore, this means that it is a legal behavior for aircraft to leave the captured space, as a legal airfield connection can lead from within the captured space to outside of it. Similarly, it is possible for aircraft not captured in a given timestep to be captured in a future timestep. Such designs emulate real aircraft behaviors, given the long-distance capabilities of aircraft to travel between cities, states, countries, and even continents and the localized view of a captured space.

8.3 – Pre-Processing Temporal Data

The pre-processing and discretization of the temporalized scenario data is similar to the process for the non-temporalized scenarios. Temporal navigational features, and features that they are temporally related to, are mapped to a presence indictor channel using the grid-based discretization approach. Only aircraft are temporally simulated for the simplicity of this initial temporalization. Thus, only aircraft and features which have some relation to the position of aircraft—just airfields—are then included in the discretized space.

Because this experiment attempts to forecast navigational feature states, note the problem previously identified in comparing regression losses for comparing *n*-dimensional arrays; it is ineffective to do so with regression loss functions that do not account for both the spatial relativities of features and the total count of features. Therefore, to make the problem feasible, aircraft are radially expanded in the model input view to provide a larger area of coverage. This better allows the NBTE Loss to then provide more meaningful values to the optimizer to fit the weights.

For this experiment, a single scenario is temporally simulated for 8 hours with 15-second intervals between steps. An example of what this space then looks like is detailed in Figure 8-1, where both a human-view and model-view of example input data is provided.

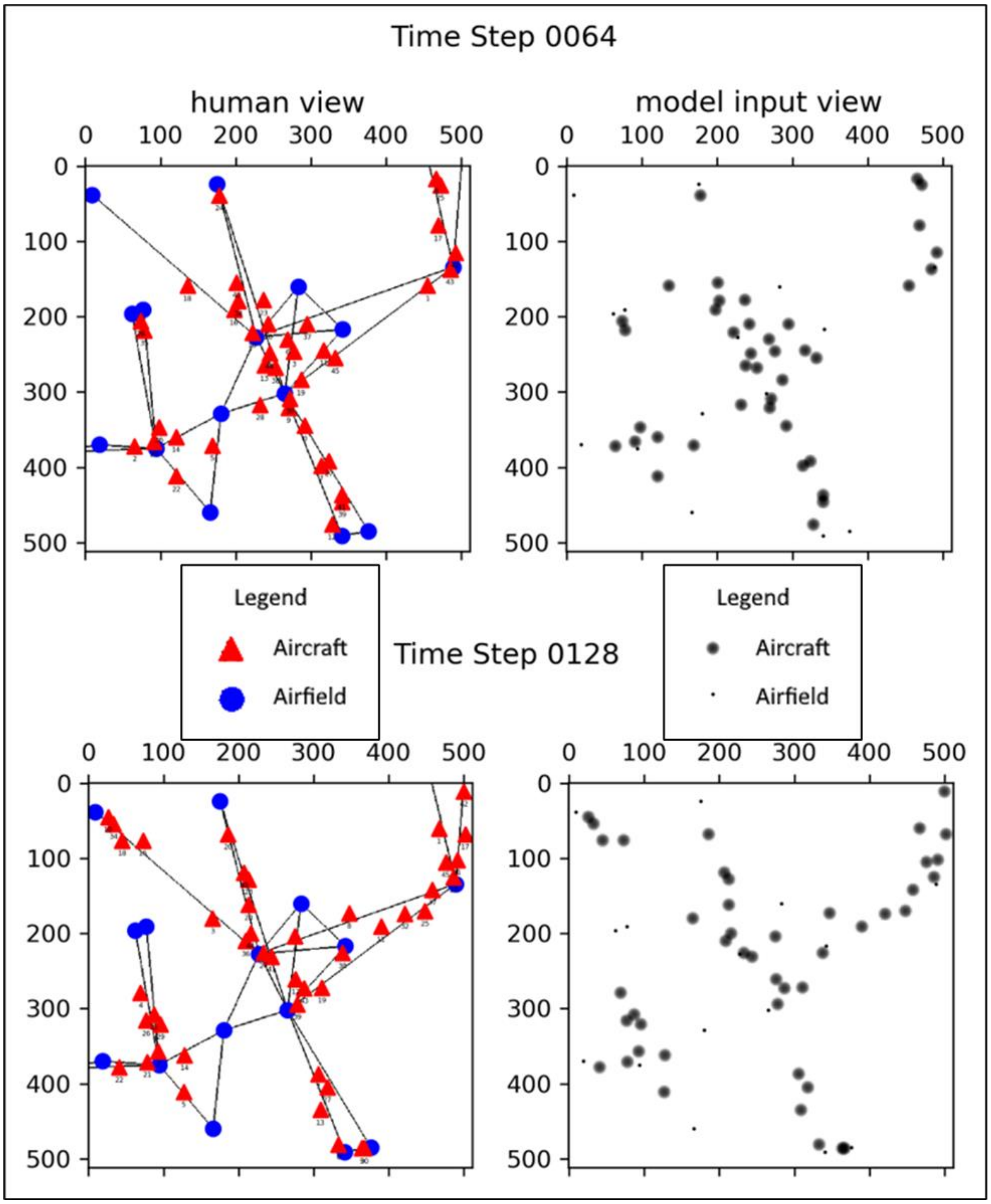

Figure 8-1 – Example Temporal Scenarios at Time Step 64 (Top) and 128 (Bottom)

8.4 – Spatial-Temporal Architecture

The architecture to model this problem must be capable of modeling spatial-temporal patterns and then provide the capability to transform navigational features as described for the spatial architectures. The architecture selected for this is the Conv-LSTM-to-Dense architecture. As the name implies, a similar layout to the Conv-to-Dense architecture is used, but instead of a series of initial CNN layers to model the navigational features of the data, a Convolution-LSTM (Long-Short Term Memory) architecture is used to model patterned relations in a historic series of input data. In some brief detail, the LSTM architecture is traditionally used to model temporal patterns in *historic* (temporally sorted) data sequences. An expansion of this architecture is the Conv-LSTM architecture, which is used to model higher dimensionality problems, such as patterns in *n*-dimensional arrays. Then, instead of producing feature maps they instead produce temporal feature maps. The Conv-LSTM-to-Dense architecture is a derivative of the Conv-to-Dense architecture in that they are structured very similarly, where the Conv-LSTM-to-Dense is effectively of a higher dimensionality. The DNN was chosen as this initial experimental output architectural style, as it can more comprehensively model features at range, whereas a TCNN is a more localized modeling style. While the added dimensionality correlates to reduced feasibility due to the exponential nature of DNNs, it was still deemed acceptable for this use-case.

8.5 – Experimental Setup

For the temporal model, training and validation data is processed in batches of 16 inputs with a sequence length of 8 timetsteps, which was found to be an effective means to train the model. Because each individual input is also 8 sequences, each batch details 128 timesteps. Batch elements are randomly sampled with replacement; the starting points for each sequence in a batch is randomly sampled. Then, the total amount of timesteps is divided by the amount of timesteps present in each batch to yield the number of steps per generation: $\frac{1920}{128} = 15$. While there is a chance for repetition of the same initial timestep, the probabilities were decided to be sufficiently low such that the marginal impact would be acceptable. From modeling 8 sequential input timesteps, the temporal model will be trained to predict the 9th.

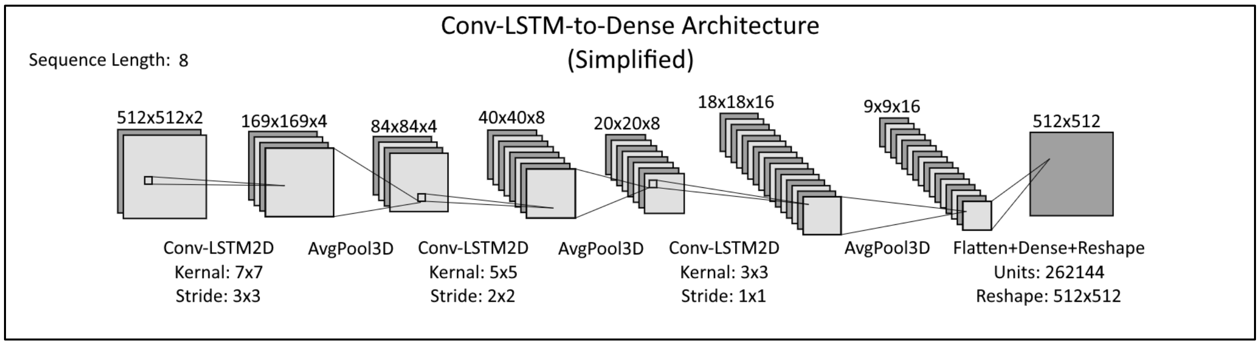

Figure 8-2 – Best Discovered Conv-LSTM-to-Dense Architecture

8.6 – Experiments and Results

These experiments attempt to demonstrate the effectiveness of the proposed temporal navigational feature modeling techniques by yielding an interpretable navigational solution that details the next timestep with respect to a given input sequence.

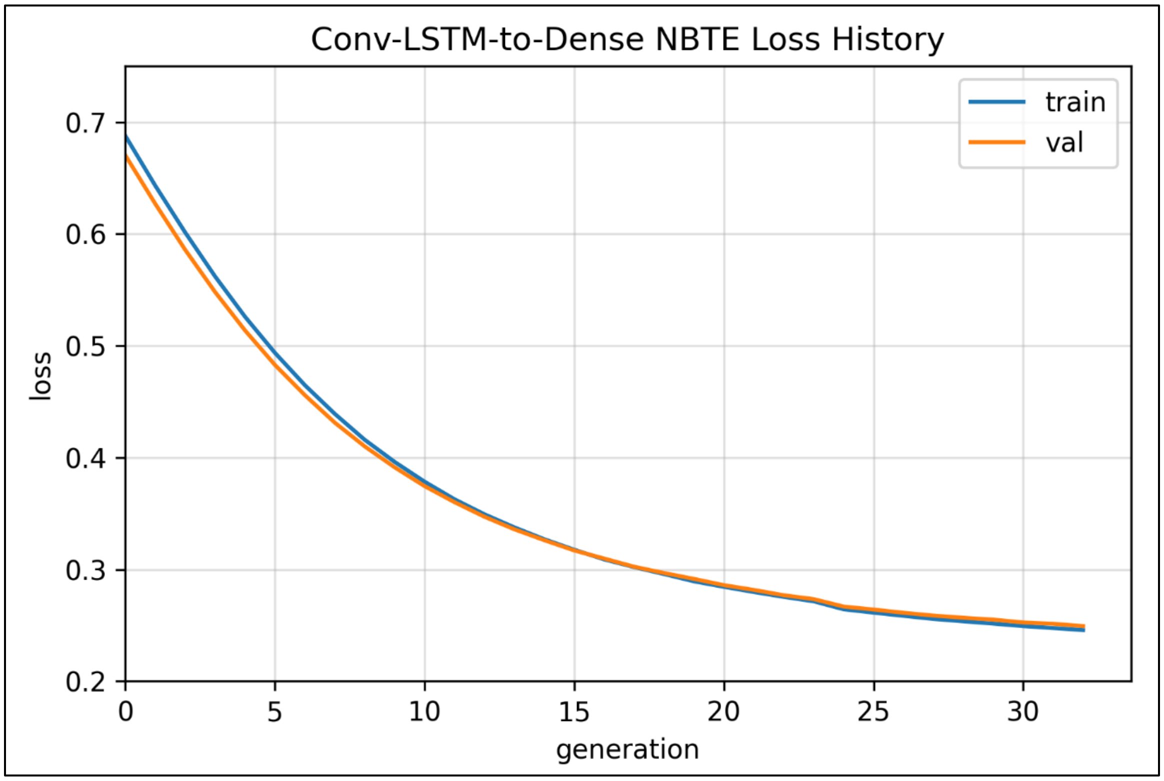

Figure 8-3 – Conv-LSTM-to-Dense NBTE Loss History

The experimental architecture instantly begins improving, however there are explosive parameter sizes that greatly impede training given the high dimensionality and measures of the space being modeled combined with the final DNN transformation. The experiment was manually halted early, utilizing the Checkpoint Callback, to observe the state of the model output once it began to reach a plateau. The predictions of the architecture are given in Figure 8-4.

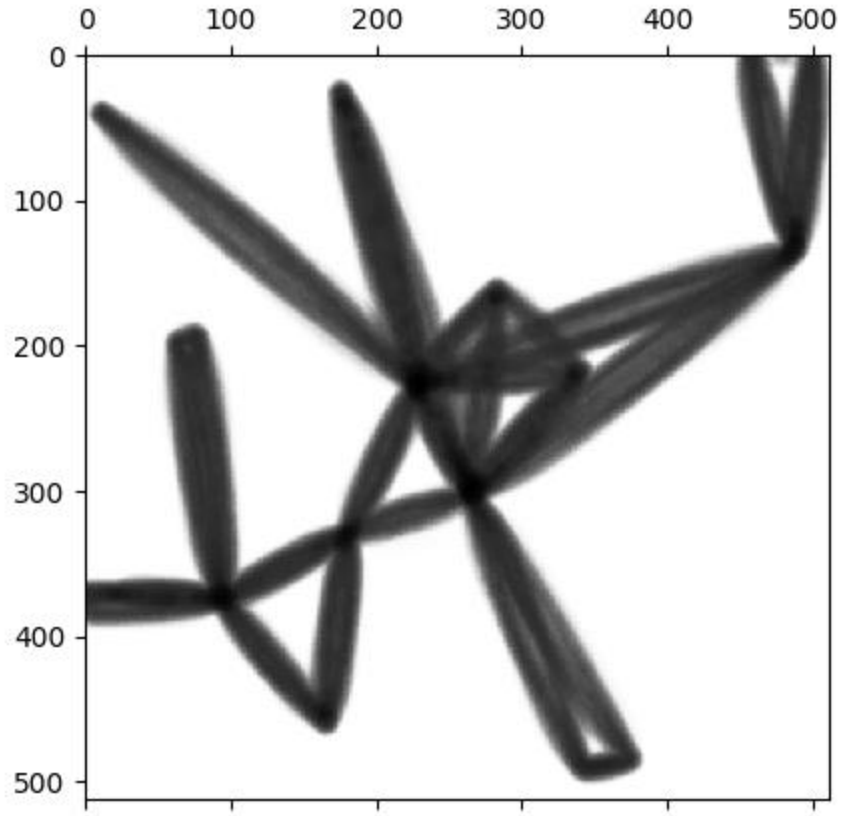

Figure 8-4 – Experimental Conv-LSTM-to-Dense Model Results Example

As the results of the trained model detail the weighted paths of travel of the temporally stimulated aircraft. Note the bowing in the line-gradients, as it identifies the confidence of an aircraft's presence in a position along the route between airfields. This is not directly a forecast of the future timestep, but is rather a generalization of risk across all timesteps of where aircraft can be positioned. The temporal architecture, in its current implementation, is decided to then be unsuccessful due to this result, useful as it may be. Furthermore, continued training was decided to be infeasible given the lengthy time requirements and how the model is beginning to plateau.

Chapter 9 Conclusions

The topic of modeling navigational features in continuous spaces is explored, where the effectiveness is empirically evaluated and demonstrated with architectural models that further transform the modeled navigational features into meaningful output in the form of navigational solutions. An airspace scenario generator is engineered and developed, which is capable of not only supporting this research but future research efforts in the domain of airspace navigation given a capability to indefinitely procedurally produce airspace scenarios that detail large variations and stochastic influences in the populating navigational features. Additionally, a new regression loss and several new measures are proposed as a means to compare and measure the correctness of positional gradients in *n*-dimensional data structures which can account for bias in the distribution of values in the structure: the *n*-Array Balanced Type Error (NBTE) Loss, Accuracy, and Coherence measures.

It is demonstrated that there exists a methodology to discretize continuous spaces and map the navigational features therein into a structure that can be analyzed by a DL architectural model, such as the Conv-to-ConvT and Conv-to-Dense architectures. It is demonstrated that the Convolutional Neural Networks (CNNs) are effective at modeling the navigational feature maps, given both the Conv-to-ConvT and Conv-to-Dense architectures detail a NBTE Accuracy of approximately 70%, despite detailing very different output stylizations. Additionally, both of these architectures detail a NBTE Coherence $\geq 56\%$, which indicates that the fitted models produce predictions distinctly different from meaningless noise.

Variations of the Conv-to-Dense architecture are also explored to determine the feasibility of capturing smaller slices of data to reduce training time and parameter counts. While both of these decreased, the performance of the model with respect to the NBTE Accuracy and Cohesion and the similarity between the distribution of predictions and the actuals is decreased. This is true for both the 128km offset tiling strategy and the 32km offset tiling strategy explored. While these results are lower than the full-resolution alternatives, they provide a still-functional methodology that can be effectively scaled for large or high-resolution navigable areas or high-resolution navigational solutions that would otherwise correspond to too many parameters to be feasible to train.

Of note in these demonstrated methodologies, it is observed that the methods used to internally model the navigational features—modeled into navigational feature maps per a number of CNN layers—is likely linked to the mechanisms by which the modeled information is transformed to the output information. This indicates that the potential for a uniform internalized means by which to model a set of navigational features is not possible for sufficiently varied architectural approaches, such as between the Conv-to-ConvT and the Conv-to-Dense architectural approaches. This is because an effective modeling strategy is dependent on how the information is transformed into an output. However, sufficiently similar architectural styles, such as the Conv-to-Dense and the Sliced Conv-to-Dense architectural approaches, may have this potential. The possibility that these navigational features are modeled differently depending on how they are transformed into an output is verified by the observed differences between the filters of different architectures, as well as the similarity of filters of similar architectures. This is a small pool of samples, with no control over the number of trained filters, which leaves much room for future investigation.

The current state of the temporal architecture is still in an initial phase of exploration, but the initial results detail some potentially useful information; it might lead to a capability to train inherent temporal features into spatial models by applying a confidence gradient of probable positions to a scenario. This would be useful in combating explosive parameter counts and would work well if the spatial modeling techniques are further improved. However, it is ultimately demonstrated that, to some degree, such an architectural model, inspired from the spatial architectural models, can potentially be adapted for temporal modeling and/or adding pseudo-temporality to spatial models.

Chapter 10 Future Work

From the contributions can be a variety of future endeavors. This thesis was motivated as helping to further the SEDA research, mentioned in §**Error! Reference source not found.**, by exploring auto-routing with DL architectures. It additionally provides a data generator that can procedurally produce indefinite amounts of interpretable data that can be used for interpretability research as human would be able to qualitatively verify the explanations of interpretable systems using this data. Additionally, in this endeavor, new statistical tools—the NBTE measures—were developed, and they can be applied to other adjacent domains as well. While described as being useful for n-dimensional arrays, they can also be applied to n-dimensional vectors given that all n-dimensional vectors can be represented as arrays with the dimensional measures: $n \times 1 \times 1 \times \ldots \times 1$, and all other permutations thereof. These three collective contributions—dataset generator, navigational solutions, and NBTE measures—can be then be the bases for multiple future works, examples of which follow.

10.1 – Improved Dataset Generation

Of notable potential is the applicability of the airspace scenario generator and the airspace scenario temporalization process that can procedurally produce large quantities of characterized data for a domain where such bulk data collection is difficult. By then improving the tools used to generate the data, models trained can be more performative in a real-world environment. Such improvements can be, for example, more detailed navigational features that more closely emulate real-world patterns or improved motion models for simulating aircraft movement.

10.2 – Alternative Architectural Approaches

Another repeated sets of experiments can also be held to observe the impact of different architectural approaches. Current architectures utilize traditional DL components, but there are other alternatives that were not initially explored. Such architectural components for modeling navigational features that could potentially be explored are ResNet (e.g., He, et al., 2015) and DenseNet (e.g., Huang, et al., 2018) Neural Networks. These are useful for solving the vanishing gradient problem that occurs when a large number of filters are used. While the useful number of filters may be dependent on the architectural used to transform the output into a navigational solution—to solve some stochastic optimization function—and thus the vanishing gradient problem does not necessarily apply, the mechanical differences in how these components internalize data could still be useful.

Currently, continuous space is discretized into a grid-based topology, which limits navigational feature modeling to CNNs and other such similar architectures. However, his produces inefficient behaviors given the biases in the space; more space is empty than not, and it can then be computationally expensive to have a multi-dimensional view of a large space. A structural alternative to the grid-based topology would be a quadtree, octree, or some n-dimension array structure where the spaces are partitioned into trees for efficient object allocations. Then alternative DL architectural components, such as Quadtree-based Convolutional Neural Network (e.g., Jayaraman, et al., 2018) (e.g., Chitta, et al., 2019), can be applied. The effectiveness of these techniques could then be compared against the results of this thesis if otherwise following similar methodologies.

Optimizers are also an integral architectural component. While the Adam optimizer is effective, there is room to experiment with other optimizers to identify any potential candidates that could be useful in the future with new architectural approaches or a direct improvement over the current even. The SGD optimizer is one potential candidate, which there is some research that show continued training with an SGD optimizer, after a model training with an Adam optimizer begins to plateau, could lead to further improvement (Keskar and Socher, 2017) (Zhou, et al., 2021).

10.3 – NBTE Similarity Measure

While the NBTE Coherence measure provides a simple and effective means to evaluate the quality of a prediction, it fails to account for the distribution of values in a multidimensional space. By calculating the similarity—statistical confidence—between two distributions, either by a Chi-Squared, Kolmogorov-Smirnov, Anderson-Darling, or other such goodness of fit tests, this can be used to scale the Neutral Accuracy variable to yield an expanded coherence measure: NBTE Similarity measure. Given some goodness of fit function h that is a function of the distribution of the predicted value Y' and the target distribution of the actual values Y, then the NBTE Similarity is given by the function $\vartheta_S(\alpha, Y)$.

$$\text{nbte similarity} = \vartheta_S(y, y', \alpha, Y, Y') \coloneqq \frac{\vartheta_A(y, y') - h(Y, Y')\vartheta_E(Y)}{1 - h(Y, Y')\vartheta_E(Y)} \tag{10.3-1}$$

Because a confidence value is in domain $(0,1)$, then the bounds of the coherence, after accounting for distribution similarity, are then also $(0,1)$. Such future work would also likely need to explore the compatibility of different goodness of fit tests with different problem spaces and distributions, and it may be determined that h could then be dependent on the type of data being evaluated. Thus, it is notated in Eq. $(10.3\text{-}1)$ as some realization of a goodness of fit test.

10.4 – NBTE Negative Coherence Uses

The range of the NBTE Coherence is given as $(-\infty, 1]$. Because $\text{coherence} = 1$ indicates maximum prediction quality and $\text{coherence} = 0$ indicates minimum quality, where a prediction is sufficiently indistinguishable from random noise, then $\text{coherence} < 0$ must inform of another property. It is conjectured then those negative coherences correlate to some potential that the complement of a prediction, with respect to the upper-bound of the domain, will be more accurate than the original prediction. Because the domain of some n-dimensional array is continuous, it is possible, however, that the complement of a prediction, along a given axis, will still incur some error. Therefore, the complement of a prediction is not guaranteed to have the complementary accuracy of the original prediction. The point at which the reliability of the complement of the prediction becomes viable is conjectured to be defined somewhere between in the exclusive domain $(-\infty, 0)$.

Training a model to reinforce negative coherence could potentially be a viable way to train a model if it is easier to model the complementary behaviors. However, given that a complement of a prediction has some likelihood to incur further errors, it is likely a more reliable and simple strategy to not take this approach. That being said, the relation and flexibility of the NBTE

Coherence would still need further exploration to conclusively determine that is the case and to more completely describe the behaviors of the NBTE methodologies.

References

A. Stringer, B. Sun, Z. Hoyt, L. Schley, D. Hougen and J. K. Antonio, "SEDA: A Self-Explaining Decision Architecture Implemented Using Deep Learning for On-Board Command and Control," 2021 IEEE/AIAA 40th Digital Avionics Systems Conference (DASC), 2021, pp. 1-10. doi: 10.1109/DASC52595.2021.9594351.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville, *Deep Learning*. Cambridge (EE. UU.): MIT Press, 2016, pp. 1-8, 12-13, 18-22, 163-220, 321-361. Available: http://www.deeplearningbook.org.

Vincent Dumoulin and Francesco Visin, “A Guide to Convolution Arithmetic for Deep Learning,” *arXiv:1603.07285 [cs, stat]*, Jan. 2018. Available: http://arxiv.org/abs/1603.07285.

Sorin Grigorescu, Brasnea Trasnea, Tiberius Cocias, and Gigel Macesanu, “A Survey of Deep Learning Techniques for Autonomous Driving,” *J. Field Robotics*, vol. 37, no. 3, pp. 362–386, Apr. 2020, doi: 10.1002/rob.21918.

Diederik P. Kingma and Jimmy Ba, “Adam: A Method for Stochastic Optimization,” *arXiv:1412.6980 [cs]*, Jan. 2017. [Online]. Available: http://arxiv.org/abs/1412.6980.

Julius Rückin, Liren Jin, and Marija Popović, “Adaptive Informative Path Planning Using Deep Reinforcement Learning for UAV-based Active Sensing,” *arXiv:2109.13570 [cs]*, Sep. 2021. [Online]. Available: http://arxiv.org/abs/2109.13570. [Accessed: Nov 2021].

Mirco Theile, Harald Bayerlein, Richard Nai, David Gesbert, and Marco Caccamo, “UAV Coverage Path Planning under Varying Power Constraints using Deep Reinforcement Learning,” *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 1444–1449, Oct. 2020, doi: 10.1109/IROS45743.2020.9340934.

Lei Tai, Giuseppe Paolo, Ming Liu, “Virtual-to-real Deep Reinforcement Learning: Continuous Control of Mobile Robots for Mapless Navigation,” *arXiv:1703.00420 [cs]*, Jul. 2017. [Online]. Available: http://arxiv.org/abs/1703.00420. [Accessed: Nov 2021].

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun, “Deep Residual Learning for Image Recognition,” arXiv:1512.03385 [cs], Dec. 2015, Accessed: Oct, 2021. [Online]. Available: http://arxiv.org/abs/1512.03385.

Gao Huang, Zhuang Liu, Laurens van der Maaten, Kilian Q. Weinberger, “Densely Connected Convolutional Networks,” *arXiv:1608.06993 [cs]*, Jan. 2018. [Online]. Available: http://arxiv.org/abs/1608.06993. [Accessed: Oct 2021].

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton, “ImageNet Classification with Deep Convolutional Neural Networks,” Commun. ACM, vol. 60, no. 6, pp. 84–90, May 2017, doi: 10.1145/3065386.

Pradeep Kumar Jayaraman, Jianhan Mei, Jianfei Cai, and Jianmin Zheng, “Quadtree Convolutional Neural Networks,” in *Computer Vision – ECCV 2018*, vol. 11210, V. Ferrari, M. Hebert, C. Sminchisescu, and Y. Weiss, Eds. Cham: Springer International Publishing, 2018, pp. 554–569. doi: 10.1007/978-3-030-01231-1_34.

Kashyap Chitta, Jose M. Alvarez, and Martial Hebert, “Quadtree Generating Networks: Efficient Hierarchical Scene Parsing with Sparse Convolutions,” *arXiv:1907.11821 [cs]*, Sep. 2019. [Online]. Available: http://arxiv.org/abs/1907.11821. [Accessed: Nov 2021].

Nitish Shirish Keskar and Richard Socher, “Improving Generalization Performance by Switching from Adam to SGD,” *arXiv:1712.07628 [cs, math]*, Dec. 2017. [Online]. Available: http://arxiv.org/abs/1712.07628. [Accessed: Oct 2021].

Pan Zhou, Jiashi Feng, Chao Ma, Caiming Xiong, Steven Hoi, Weinan E, “Towards Theoretically Understanding Why SGD Generalizes Better Than ADAM in Deep Learning,” *arXiv:2010.05627 [cs, math, stat]*, Nov. 2021. [Online]. Available: http://arxiv.org/abs/2010.05627. [Accessed: Nov 2021].