


8-2022

Deep Learning Applications in Industrial and Systems Engineering

Winthrop Harvey
University of Arkansas, Fayetteville

Follow this and additional works at: <https://scholarworks.uark.edu/etd>

 Part of the [Artificial Intelligence and Robotics Commons](#), [Industrial Engineering Commons](#), [Remote Sensing Commons](#), and the [Systems Engineering Commons](#)

Citation

Harvey, W. (2022). Deep Learning Applications in Industrial and Systems Engineering. *Graduate Theses and Dissertations* Retrieved from <https://scholarworks.uark.edu/etd/4671>

This Dissertation is brought to you for free and open access by ScholarWorks@UARK. It has been accepted for inclusion in Graduate Theses and Dissertations by an authorized administrator of ScholarWorks@UARK. For more information, please contact scholar@uark.edu, uarepos@uark.edu.

Deep Learning Applications in Industrial and Systems Engineering

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy in Industrial Engineering

by

Winthrop Harvey
Amherst College
Bachelor of Science in Mathematics, 2013

August 2022
University of Arkansas

This dissertation is approved for recommendation to the Graduate Council.

Chase Rainwater, Ph.D.
Committee Chair:

Ed Pohl, Ph.D.
Committee Member

Jackson Cothren, Ph.D.
Committee Member

Xiao Liu, Ph.D.
Committee Member

Abstract

Deep learning - the use of large neural networks to perform machine learning - has transformed the world. As the capabilities of deep models continue to grow, deep learning is becoming an increasingly valuable and practical tool for industrial engineering. With its wide applicability, deep learning can be turned to many industrial engineering tasks, including optimization, heuristic search, and functional approximation. In this dissertation, the major concepts and paradigms of deep learning are reviewed, and three industrial engineering projects applying these methods are described. The first applies a deep convolutional network to the task of absolute aerial geolocalization - the regression of real geographic coordinates from aerial photos - showing promising results. Next, continuing on this work, the features and characteristics of the deep aerial geolocalization model are further studied, with implications for future applications and methodological improvements. Lastly, a deep learning model is developed and applied to a difficult rare event problem of predicting failure times in oil and natural gas wells from process and site data. Practical details of applying deep learning to this sort of data are discussed, and methodological principles are proposed.

ACKNOWLEDGEMENTS

Thank you to the University of Arkansas and the department of Industrial Engineering for being a home for these last four years and making this possible. A special thanks to my advisor, Chase Rainwater, for his mentorship, guidance, and understanding.

TABLE OF CONTENTS

Abstract	1
Table of Contents	1
1 Introduction	1
1.1 Deep Learning in Industrial and Systems Engineering	1
2 A Brief Overview of Modern Deep Learning: Methods and Concepts	3
2.0.1 Deep Neural Network Overview	3
2.0.2 Important Neural Network Concepts	7
2.0.3 Major Neural Network Architecture Types	11
2.0.4 Closing Remarks	14
3 Direct Aerial Visual Geolocalization Using Deep Neural Networks	16
3.1 Introduction	17
3.2 Methods and Results	23
3.2.1 Data Choice and Processing	23
3.2.2 Network Architecture Selection	25
3.2.3 Field of View Comparison	27
3.2.4 Loss Function Comparison	27
3.2.5 Training Replicability	29
3.2.6 AVL Results	29
3.2.7 Uncertainty Calibration	30
3.3 Discussion	32
3.3.1 Accuracy and Loss Function Choice	32
3.3.2 Uncertainty Calibration	34
3.3.3 Computational Considerations	35
3.3.4 Input and Output Choices	35
3.3.5 Future Directions for AVL Deep Models	37
3.4 Conclusions	39
3.5 Appendix	42
3.5.1 Neural Network Implementation Details	42
3.5.2 Data Set Characteristics	43
4 Expanding Deep Learning for AVL	47
4.1 Introduction	47
4.1.1 Methods Applicable to All Experiments	47
4.1.2 Interpreting Results Between Experiments	48
4.2 Network Architecture and Training Approaches	49
4.2.1 Introduction	49
4.2.2 Methods	50
4.2.3 Results and Discussion	52
4.3 Network Parameter and Region of Interest Scaling/Specificity	55
4.3.1 Introduction	55
4.3.2 Methods and Results	56
4.3.3 Discussion	58
4.4 Training Data Requirement	60
4.4.1 Methods and Results	61
4.4.2 Discussion	61
4.5 Model Distribution Calibration	62
4.5.1 Introduction	62
4.5.2 Methods and Results	66
4.5.3 Discussion	68
4.6 Occlusion and Orientation Error	71
4.6.1 Introduction	71
4.6.2 Methods	72
4.6.3 Results	72
4.7 Direct Raster Training Pipeline	75

4.7.1	Methods, Results, and Discussion	75
5	Deep Learning for Repairable System Reliability Prediction	78
5.1	Problem Introduction	78
5.2	Deep learning approaches to tabular data literature	79
5.2.1	General Tabular Data Problems	79
5.2.2	Deep Learning On Time-Series Data	82
5.2.3	Deep Learning Approaches to Rare-Event Problems	83
5.2.4	Non-Deep Approaches	85
5.2.5	Deep Learning Approaches to Repairable Systems Literature	86
5.3	Methods Introduction	86
5.3.1	Feature/Label Considerations	86
5.3.2	Architecture	88
5.3.3	Feature Embedding	90
5.3.4	Regularization	91
5.3.5	Measuring Model Performance	91
5.3.6	Class Balancing	92
5.4	Methods Specification	93
5.5	Results	95
5.6	Discussion	96
5.6.1	Model Performance	96
5.6.2	Data Troubles - Is the problem possible?	97
5.6.3	Future Work	102
5.6.4	Conclusion	102
5.6.5	Acknowledgements	103
6	Conclusion	105

Published Works in this Dissertation

Chapter 3, published 2021, cited as: Harvey W, Rainwater C, Cothren J. Direct Aerial Visual Geolocation Using Deep Neural Networks. *Remote Sensing*. 2021; 13(19):4017. <https://doi.org/10.3390/rs13194017>

1 Introduction

1.1 Deep Learning in Industrial and Systems Engineering

Over the last decade machine learning has come to occupy an increasingly important role in our society, with broad applications in diverse fields of human endeavor. One of the major drivers of this increased importance has been the continual development and success of deep neural networks (DNNs). Much of the research and development of these algorithms has come from the field of computer science. However, algorithms and heuristic methods have long been a major field in industrial and systems engineering. Accordingly, researching improvements and novel applications of DNNs in the context of industrial engineering has already begun to come into its own as a major field of focus. In this document, the use of newly developed machine learning techniques is investigated in the context of several different systems engineering problems. It comprises a body of work contributing effective new approaches, and proposes additional research to further advance each application.

Chapter 2 establishes some of the terminology and methodology of modern deep learning, introducing concepts and tools that are used throughout the remaining chapters.

Chapter 3 discusses the development of a convolutional neural network (CNN) based approach to the task of Absolute Visual Geolocalization (AVL). AVL, or frame-to-reference geolocalization, compares current visual data from an aircraft to a trusted geographic reference in order to determine the aircraft's current location. Compared to relative or frame-to-frame methods of localization, AVL is not susceptible to drift in location estimate. Neural networks are particularly attractive for this task because of their robustness to differences between input images and reference images, as well as their computational efficiency. This efficiency is of particular importance for AVL applications on unmanned aerial vehicles where resources are often limited. This work is complete and was recently published[2] in the journal Remote Sensing as "Direct Aerial Visual Geolocalization Using Deep Neural Networks."

Chapter 4 extends and expands the work presented in Chapter 3. Organized into six groups of experiments, this chapter dives deep into the questions and hypotheses that the work in Chapter 3 introduces. Model performance is tested under a variety of configurations, leading to improved understanding of what characteristics of a deep learning model are

important to performance. Furthermore, methodological improvements are developed and tested to improve model calibration and the ease of data preprocessing.

Chapter 5 discusses the investigation of DNN based methods on a data set of failure times of 8,232 oil and natural gas wells. Liu and Pan [3] first presented this data set, alongside a novel algorithm RF-R (random forests for repairable system reliability analysis) for handling the dataset’s large size and special characteristics. Although DNN based approaches have shown large gains over traditional machine learning approaches in fields such as computer vision and natural language processing, their relative performance on tabular data is poorer. This data set, which is also time series based and is an example of a rare-event problem (since failures are rare compared to non-failures), is particularly challenging for standard DNN approaches. Tabular data is incredibly important and makes up the majority of business data sets. The continued and growing demand for better ML approaches to tabular data is highlighted by the popularity of applications such as Google’s AutoML Tables[1]. Time series and rare-event problems are also commonly encountered. Thus, the development of improved methods of approaching these types of problems can provide substantial value in many contexts beyond failure detection in oil and natural gas wells. This project is part of the Arkansas DART (Data Analytics that are Robust and Trusted) initiative.

References

- [1] E. Bisong. “Google AutoML: Cloud Natural Language Processing”. In: *Building Machine Learning and Deep Learning Models on Google Cloud Platform*. Springer, 2019, pp. 599–612.
- [2] W. Harvey, C. Rainwater, and J. Cothren. “Direct Aerial Visual Geolocalization Using Deep Neural Networks”. In: *Remote Sensing* 13.19 (2021), p. 4017.
- [3] X. Liu and R. Pan. “Analysis of large heterogeneous repairable system reliability data with static system attributes and dynamic sensor measurement in big data environment”. In: *Technometrics* 62.2 (2020), pp. 206–222.

2 A Brief Overview of Modern Deep Learning: Methods and Concepts

2.0.1 Deep Neural Network Overview

Deep learning is a subset of machine learning using deep neural networks. It is a heuristic¹ functional approximation procedure that is widely applicable and produces state-of-the-art results over a wide variety of problem domains. It is a heavily data driven approach known for producing extraordinary results given large enough data sets to train on. Already, deep learning has transformed many industries. As our world becomes increasingly data driven, the influence of deep learning is only going to increase, and the demand for knowledgeable practitioners and researchers with it. As a type of heuristic algorithm that is broadly applicable to many systems and industrial applications, deep learning is becoming a leading tool for many systems engineers and a frequent topic of operations research. In this chapter, basic concepts, definitions, and equations that underlie deep learning are presented.

Artificial neural networks (ANNs) produce outputs from some (fixed) number of inputs by progressing through a network of computational nodes (also called neurons or hidden units). At each node, the input has some computation performed on it (the specific operation is determined by the node's type which is designed in advance), and the result is then multiplied by trainable weights (also known as a parameters) before serving as input for other nodes in the next layer. The output from each layer forms the input for the next, until the output layer is reached. The term “deep” refers to the fact that in most modern neural networks there are a large number of nodes and layers. The goal of the training process is to find a set of weights which closely approximates the true function relating inputs to outputs over the entire input domain. In equation form, we are seeking weights w such that the overall function defined by the neural network, $\hat{f}(x;w)$, closely approximates some target function $f(x)$ over the set of all inputs x in the domain. That is, for a feedforward neural network, we want to find w such that

¹Although there are some theoretical guarantees on model convergences under for specific architectures (e.g. a multi-layer perceptron under plain gradient descent with no regularizers or other complicating factors) and long timespans, in practice most deep learning applications are heuristic methods

$$\hat{f}(x; w) \approx f(x)$$

for all x in the domain.

The simplest deep neural network architecture, known as a multilayer perceptron, consists of several layers of nodes where each node is connected to every node in the next layer (these layers are therefore also known as dense layers). Each connection has an associated weight, as described above, which multiplies the output. Each node also (usually) has a bias weight which directly adds to its input. Finally, each node has an associated activation function which also modifies its output.

In equation form, the input to a node, z , with n nodes in the previous layer is

$$z = b + \sum_{i=1}^n o_i * w_i$$

where o_i is the output associated with the i^{th} node in the prior layer, and w_i is the weight from the i^{th} node to the current node. This input is then modified by the nodes activation function to serve as that node's output into the next layer. Any almost-everywhere differentiable function can be used as an activation function. Typically, a simple nonlinear activation function is used. The nonlinearity of the activation function is important for being able to approximate nonlinear functions, as will be discussed shortly. By way of example, one of the most commonly used activation functions in deep neural networks is the ReLU (Rectified Linear Unit) function:

$$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$$

which simply makes the output zero if the input is below 0, and otherwise is the identity function.

The importance of having nonlinear activation functions, as well as the fundamental property that makes neural networks effective, comes from their ability to serve as universal function approximators. It has been proven [6] that multilayer feedforward neural networks are capable of approximating any function to arbitrary precision, provided that they use

nonlinear activation functions, and are of sufficient size (have enough parameters).

Given that any function can be approximated by a neural network, the next question is how to find the weights that best approximate the desired function. A simple but effective method of is to perform gradient descent relative to a loss function. The loss function should be minimized when the function defined by the neural network matches the target function - in other words, when the output of the neural network matches the target label. For regression problems, the loss function can be an error measure such as mean squared error. Next, we compute the gradient of the loss relative to every weight in the network through a chain differentiation procedure known as backpropagation. Then taking a small step in the direction of loss reduction, the neural network will reduce its error for that input-output pair, as that input should now produce a reduced loss. The update rule for backpropagation for a specific weight w is:

$$w_{new} = w_{old} - \eta \frac{\partial L}{\partial w}$$

where L is the loss, w_{new} is the updated weight, w_{old} is the old weight, and η is a constant called the learning rate which controls how large the gradient descent steps are. The partial derivatives for the weights of the output layer are immediately calculable using the derivative of the activation function and the weight. From these derivatives, the derivatives of the next interior layer can be calculated in a similar manner using the chain rule, thereby propagating the loss backwards (backpropagation). In practice, it is not necessary to explicitly calculate the derivatives of activation functions as modern deep learning libraries use automatic differentiation.

Performing backpropagation for one input will reduce the loss on that input. However, the goal is to minimize the loss across all possible inputs in the input domain. If we had the capability to train the network on all possible inputs, we could train it until we were sure it was highly correct on every one. In real-world problems where we cannot present all possible inputs, we instead settle for exposing the neural network to as broad a variety of input-output pairs as possible, so that it gradually learns a generalized functional relationship that approximates the target function. This requires knowledge of the output label for each input (labeled data), and so is a type of supervised learning. One iteration over all available training data is called an epoch. The generalization capability of a neural network is usually

evaluated on a test set, which is simply a set of data from the input domain that was not used during training. This process of updating the weights for each presentation of input using backpropagation is known as stochastic gradient descent (SGD). It is stochastic because each individual weight update may not be beneficial. However, in practice, over time networks trained using SGD usually progress towards better performance. Many alternatives to SGD have been developed, usually adapting gradients on specific weights based on the history of the training process and heuristic rules. A popular and successful example is Adam [8], which modifies the learning rate of each weight based on estimates of their first and second moments.

One important note about loss functions is that they are often not the metric we are trying to optimize towards. For example, if we want to maximize the accuracy of a classifier, we can't directly use accuracy as a loss function as accuracy only has meaning over sets of inputs, and therefore can't be calculated/differentiated from a single input sample. Even when the desired metric for optimization can be used directly as a loss function, it will not necessarily achieve the best results. This is because a neural network does not simply optimize weights relative to a loss function. Rather, a neural network uses the loss function as part of a highly stochastic process to navigate a complicated high-dimensional loss surface. As an example, in reinforcement learning direct policy gradient methods are unstable and tend to have poor performance². Proximal Policy Optimization [10], which uses a modified, clipped form of the policy gradient loss function, is highly successful. The significant take-away is that a modified form of the loss function can achieve better performance than the original loss function even as measured by the original function. This is a major reason deep learning is better thought of as a heuristic approach than a type of optimization method.

² This is because of the reciprocal dependence of actions determining the feedback from the environment, while the feedback from the environment determines the actions to be taken, leading to vicious cycles where the agent becomes stuck in a local optimum, but because it is stuck does not get feedback enabling it to learn to take the actions that would allow it to leave the local optimum

2.0.2 Important Neural Network Concepts

Regularization

A universal functional strategy that works on any finite input set is a lookup table (memorization). Since input data sets are finite in practice, this is a major problem for neural networks. Memorization allows for perfect accuracy on any input in the training set. However, it does not generalize at all beyond this. Since the goal of deep learning applications is to learn a general functional relationship that works over the entire input domain, including those inputs that were not used in training, it is important to take steps to encourage generalization. The terminology for the phenomenon of a neural network overspecializing on its training set to the detriment of performance on data in the test set is known as overfitting. Collectively, methodologies to reduce overfitting are known as regularization methods.

Some of the most important regularization techniques are parameter reduction, early stopping, weight decay, batch normalization, and dropout.

- Parameter reduction is restricting the size of the neural network so that its expressivity (representational capacity) is reduced. For simple problems, few hidden units are needed to effectively capture the relationship between input and output. If the network is larger than required, the extra capacity is usually used to memorize specific inputs. Thus, a method to reduce overfitting is to ensure that the neural network used is not larger than necessary for the problem. This also saves on computation.
- Early stopping is ending training of a network when it is noticed that the network has begun overfitting, and test set performance is dropping. Networks often learn general relationships first and then begin to learn more specific memorization based strategies later, so this is a common pattern to encounter.
- Weight decay adds a penalty term to network weights encouraging them to remain closer to zero. This favors simpler functional relationships which are more likely to generalize. Memorization strategies are very parameter intensive due to their high informational requirements. There are many different types of weight decay penalties

that can be applied at varying intensities, with different mathematical properties and effects on network training.

- Batch normalization rescales and recenters batches of inputs at internal points in the network. The exact reason for batch normalization's effectiveness as a regularization technique are not clear and this is still a matter of active research and debate. However, batch normalization is often one of the most effective regularization techniques in practice and is widely used.
- Dropout randomly excludes certain nodes during each training step. This encourages robustness and discourages memorization as the network must learn to be accurate even using a random subset of its nodes. Additionally, it is unlikely to have the same nodes active when presented with the same input again on subsequent epochs. Conceptually, this turns a single network into an ensemble of networks during training.

Hyperparameter Tuning

In addition to the parameters that are determined by training (the weights), neural networks also have a large number of parameters that are not trained but rather determined ahead of time by the practitioner. These are called hyperparameters. Examples of hyperparameters include the number of layers in a network, the number of hidden units per layer, the choice of activation function, the learning rate and its evolution over time, the choice of learning algorithm, parameter levels of various regularization techniques like weight decay, levels and types of data augmentation and their associated parameters, how input data is preprocessed and presented, the train/test split, and even the random seed. This is only a small sample of the number of possible hyperparameters - and each represents a decision that a deep learning practitioner must make. Unfortunately, there are largely no a priori way to determine optimal hyperparameter settings. Therefore, practitioners must rely on rules-of-thumb and empirical hyperparameter optimization techniques. Fortunately, many such rules and hyperparameter optimization methods have emerged. Finding more efficient ways to set hyperparameters remains a major area of research, as hyperparameter tuning can be highly time and resource intensive.

Data Preprocessing

Neural networks are fundamentally mathematical constructs and are sensitive to the numerical properties of the input data. As a simple example, suppose you were to apply a neural network to a problem of predicting three different quantities, and you used mean-squared error as the loss functions. If one of the three quantities had a much larger scale than the others, it would also generate more loss, and the majority of learning in the network would go towards optimizing estimates of this one quantity even at the expense of the other two. Another common issue arises with large categorical inputs, where simple encoding techniques may cause issues during training by generating large sparse inputs. Because most architectures can only accept fixed size inputs, they struggle heavily with missing data. A neural network can't accept non-numerical input, so any record with missing data must either be removed or processed to assign the missing data some numerical value (the choice of which can introduce bias). Because of issues like this, data inputs into neural networks must undergo careful preprocessing.

Data Augmentation

Deep learning approaches are data hungry. However, in most real-world problems the amount of data that is available or is feasible to gather is limited. Consequently, methods of data augmentation, where additional unique input samples are generated, are important in many deep learning projects. Typically, data augmentation techniques work by observing some transformation of the available data that changes it, but does not affect the label (or affects it in some known way). An example would be turning a picture of a dog upside-down. It is still a picture of a dog, but nearly every pixel value has changed. Additionally, rotation is a reasonable transformation that might occur on real pictures of dogs. A transformation should be plausible and relationship preserving in order to be considered as a candidate for data augmentation.

Unsupervised and Semi-supervised Learning

The discussion above has assumed that the training data consists of input-output pairs with known labels, thereby allowing supervised learning. However, neural networks

are also highly capable in unsupervised (where there are no labels, as in autoencoders that reproduce input), and semi-supervised tasks (where some labeled data serves as a starting point to bootstrap classification on a larger number of unlabeled samples). These techniques can reach impressive accuracies on extremely limited amounts of labeled data. For example, Albert et al. achieved error rates of only 8.46% on the CIFAR-10 dataset using just one (carefully chosen) labeled datum per classification category.

Computational and Training Efficiency

Because of the very large number of training examples that need to be presented to train a neural network, as well as the large number of parameters for many state of the art models, computational and training efficiency are a major area of concern. Once fully trained, neural networks are very fast at doing only inference (no backpropagation). Computational efficiency can be further improved by a variety of methods, such as parameter pruning where weights that are not influential (e.g. they have value zero and so carry no information) are removed from the network.

Neural networks typically require a large amount of matrix operations, which can be parallelized. In general, we want to exploit parallelization as much as possible by increasing the number of training samples presented during each training step (batch size). However, care should be taken that batch size does not become large relative to the data set, as this can create additional problems. Graphics/Tensor Processing Units are hardware that are specialized for highly parallel matrix operations and are therefore extremely effective at training and running neural networks. The increased demand for deep learning powered applications in portable contexts has led to the development of mobile GPUs/TPUs that can fit into even small devices, allowing deep models to be deployed more widely than ever before.

For training efficiently, we want to avoid bottlenecks where the GPU is idle due to waiting for input data. Thus, data preprocessing and loading pipelines must be designed to be fast enough to always have data ready for the GPU. As GPUs have increased in speed tremendously, it is increasingly the case that training speed is limited by CPU bound data preprocessing or data access. To avoid this, techniques such as data preloading (preparing

and caching the next set of data inputs while the GPU is working), and offline preprocessing/caching should be used.

Interpretability

Deep neural networks are highly complex with very large numbers of parameters. This makes the internal operation of the network difficult to analyze, leading to a problem of interpretability - what some call the "black box" problem. You can use the box, but you can't see inside it to see how it works. As deep learning plays an increasingly large role in our society, the need for accountability and explainability in neural networks has also grown. To this end, an entire field of neural network interpretability research has emerged. A popular example is the method of integrated gradients [11], which computes the effect of inputs on outputs relative to a baseline input. This gives a better picture of which inputs were most influential in causing the output. Another technique is to include probability distributions including estimates of uncertainty explicitly in the network's output.

2.0.3 Major Neural Network Architecture Types

In addition to the basic multilayer perceptron, there are many other types of neural network architectures that have been developed with various specialties. I will briefly cover some of the major archetypes that see use today.

Convolutional Neural Networks

Convolutional neural networks (see [9] for the first major paper applying backpropagation trained CNNs to image processing) use convolutional layers to process input, typically images. Inspired by image processing techniques, these convolutional layers have kernels (also known as filters) which translate over the input producing output at each step. This enables efficient image processing as filters can learn to respond to input no matter where it is in the image. It also improves computational efficiency relative to fully dense networks as it has far fewer parameters to train. The fundamental insight behind CNN's success is that nearby pixels have far more inter-relationships than distant pixels. However, in a dense network processing image input, every single pixel is equally connected to every other pixel in terms of the network architecture. By doing away with many of the superfluous weights,

CNNs are able to far more effectively process image data. CNNs form the backbone of almost every major deep learning computer vision application today and demonstrate the strength of incorporating domain knowledge into the model architecture.

Residual Neural Networks

As networks become larger and deeper, they become more difficult to train not only because of the increase in the dimensionality, but also because the successive modification of gradients by many layers often leads to either vanishing or exploding gradients: gradients that become very close to 0 or very large. Very small gradients lead to stagnating training, whereas very large gradients lead to high instability. Residual Neural Networks [4] address this problem through the presence of skip connections which bring inputs (and therefore gradients) directly from early layers to later layers skipping the layers in-between. This method has proven to be extremely effective and skip connections are now a common feature in many types of deep neural networks. Convolutional neural networks, as well as other types, can easily incorporate skip connections and also be a residual neural network.

Recurrent Neural Networks

Unlike feedforward neural networks where all connections move from input to output, recurrent (or stateful) neural networks also have connections backward. Since this creates loops in the computational graph, to avoid an infinite regress these backwards connections typically only output once per training step, with their output based on the previous input(s). What this causes is a dependence in the output not only on the current input, but also the sequence of inputs before it. This is desirable for sequence data such as time-series data or in natural language processing. A famous example is the Long Short-Term Memory [5] network, which is a recurrent neural network that also learns to retain or forget certain information.

Recurrent neural networks can be very effective, but they have a major weakness: the dependence of the output on the sequence of inputs means that training cannot be fully parallelized. Every input sequence must be handled in order. Recently, attention-based networks such as Transformer models have shown great success on sequence data while

still allowing for parallelization, leading to a shift away from recurrent neural networks. However, LSTM networks still maintain advantages over Transformers on very long sequences (Transformers are not stateful and can't maintain context outside of their input window) and on smaller datasets (Transformers are considered more data hungry).

Generative Adversarial Networks

In a Generative Adversarial Network [2], two networks are trained. One, the generator, must create convincing output from a latent distribution (typically Gaussian noise) in order to fool a discriminator. The other, the discriminator, must learn to determine when it is presented with a real sample and when it is presented with a sample generated by the other network. Ideally, the two networks train together until the generator can produce output that is extremely difficult to distinguish from the real-world training data. This can be used to, for instance, generate convincing photographs of people who do not actually exist. GANs can be highly effective, but the min-max nature of the training problem can lead to high instability and problems such as mode collapse where the two networks hyperspecialize against each other, but fail to achieve the goal of diverse output. As an example for how the min-max problem can lead to training instability, consider what happens when the discriminator becomes too effective against the generator, a common problem early in training. If every output the generator is capable of producing is always identified as generated, the generator gets the same feedback on everything it produces and fails to learn which attempts are closer to being successful. Training GANs is a careful balancing act, but they can produce incredible results at their best. Most deep learning image generation applications are powered at some level by GANs.

Attention-Based Networks

Attention-Based Networks learn to "attend" to different parts of the input based on context. This attention is not just numerical, but computational. Input that garners attention undergoes more processing, while input that is not undergoes less. The network learns where to prioritize its own computational resources. At this time, Attention-Based Networks are practically synonymous with Transformer-based networks [12] which dominate

the field. Transformer based architecture have vastly expanded the capabilities of natural language processing deep models and have also shown state-of-the-art capabilities on a variety of other tasks. Transformer architectures are highly parallelizable, making them quick to train relative to other models that achieve similar performance.

Transformer based models are an incredibly exciting development as they seem to be a major step forward for neural network on a variety of different tasks. A major thrust of deep learning research at the current time is applying Transformer based architectures to novel tasks and improving Transformer architectures to address their weaknesses. Despite their state-of-the-art performance, default Transformer networks do have some large downsides. First, the computational and memory requirements grow quadratically with input size, limiting the size of the input window, especially in applications where hardware resources are limited. Transformer models are also generally slower at inference than LSTM based models. Transformer models are not stateful, so they cannot maintain context over long sequences (though see [7][1] for examples of introducing statefulness to Transformer model for long input sequences). Transformer models are also considered to be very data hungry (even relative to other deep models) and difficult to train, though approaches are being developed to alleviate this, see [3] for an example.

2.0.4 Closing Remarks

The common thread across my work has been the development and implementation of deep models. In the remaining chapters, concepts and tools presented in this chapter are used in research projects involving two very different tasks. In Chapter 3, the application of CNNs to direct Absolute Visual Localization are investigated. This research takes an unusual approach of using a distributional layer with negative log-likelihood as the loss function and achieve good results that demonstrate the feasibility of direct AVL using the approach. In Chapter 4, the models and ideas introduced in Chapter 3 are further developed and explored, leading to deeper understanding of how deep learning can best be leveraged for AVL tasks. In Chapter 5, a deep learning approach for a case study problem whose characteristics make it particularly challenging for deep approaches is developed. The chapter offers a model blueprint based on principles of data preprocessing, CNN architecture, data augmentation,

and data regularization to address the special characteristics of the case study.

References

- [1] Z. Dai, Z. Yang, Y. Yang, J. Carbonell, Q. V. Le, and R. Salakhutdinov. “Transformer-xl: Attentive language models beyond a fixed-length context”. In: *arXiv preprint arXiv:1901.02860* (2019).
- [2] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. *Generative Adversarial Networks*. 2014. arXiv: 1406.2661 [stat.ML].
- [3] A. Hassani, S. Walton, N. Shah, A. Abuduweili, J. Li, and H. Shi. “Escaping the big data paradigm with compact transformers”. In: *arXiv preprint arXiv:2104.05704* (2021).
- [4] K. He, X. Zhang, S. Ren, and J. Sun. “Deep residual learning for image recognition”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.
- [5] S. Hochreiter and J. Schmidhuber. “Long short-term memory”. In: *Neural computation* 9.8 (1997), pp. 1735–1780.
- [6] K. Hornik. “Approximation capabilities of multilayer feedforward networks”. In: *Neural networks* 4.2 (1991), pp. 251–257.
- [7] A. Katharopoulos, A. Vyas, N. Pappas, and F. Fleuret. “Transformers are rnns: Fast autoregressive transformers with linear attention”. In: *International Conference on Machine Learning*. PMLR. 2020, pp. 5156–5165.
- [8] D. P. Kingma and J. Ba. “Adam: A method for stochastic optimization”. In: *arXiv preprint arXiv:1412.6980* (2014).
- [9] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. “Backpropagation applied to handwritten zip code recognition”. In: *Neural computation* 1.4 (1989), pp. 541–551.
- [10] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. “Proximal policy optimization algorithms”. In: *arXiv preprint arXiv:1707.06347* (2017).
- [11] M. Sundararajan, A. Taly, and Q. Yan. “Axiomatic attribution for deep networks”. In: *International Conference on Machine Learning*. PMLR. 2017, pp. 3319–3328.
- [12] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. “Attention is all you need”. In: *Advances in neural information processing systems*. 2017, pp. 5998–6008.

Direct Aerial Visual Geolocalization Using Deep Neural Networks

Winthrop Harvey ¹, Chase Rainwater ¹ and Jackson Cothren ²

¹ Department of Industrial Engineering, University of Arkansas, Fayetteville

² Department of Geosciences, University of Arkansas, Fayetteville

Abstract

Unmanned aerial vehicles (UAVs) must keep track of their location in order to maintain flight plans. Currently, this task is almost entirely performed by a combination of Inertial Measurement Units (IMUs) and reference to GNSS (Global Navigation Satellite System). Navigation by GNSS, however, is not always reliable, due to various causes both natural (reflection and blockage from objects, technical fault, inclement weather) and artificial (GPS spoofing and denial). In such GPS-denied situations, it is desirable to have additional methods for aerial geolocalization. One such method is visual geolocalization, where aircraft use their ground facing cameras to localize and navigate. The state of the art in many ground-level image processing tasks involve the use of Convolutional Neural Networks (CNNs). We present here a study of how effectively a modern CNN designed for visual classification can be applied to the problem of Absolute Visual Geolocalization (AVL, localization without a prior location estimate). An Xception based architecture is trained from scratch over a >1000 km² section of Washington County, Arkansas to directly regress latitude and longitude from images from different orthorectified high-altitude survey flights. It achieves average localization accuracy on unseen image sets over the same region from different years and seasons with as low as 115 meters average error, which localizes to .004% of the training area, or about 8% of the width of the 1.5x1.5km input image. This demonstrates that CNNs are expressive enough to encode robust landscape information for geolocalization over large geographic areas. Furthermore, discussed are methods of providing uncertainty for CNN regression outputs, and future areas of potential improvement for use of deep neural networks in visual geolocalization.

This work was published as: Harvey, W., Rainwater, C., & Cothren, J. (2021). Direct Aerial Visual Geolocalization Using Deep Neural Networks. *Remote Sensing*, 13(19), 4017.

3.1 Introduction

Pilotage or piloting is the practice of using vision to navigate an aircraft, usually by reference to terrain and landmarks. This is in contrast to flying by instrument. Modern aircraft instruments enable navigation even in the absence of vision. However, vision and visual flight remains immensely valuable to pilots when it is available, both in improving navigation quality and as a check against instrument error. Despite possessing cameras, automated drones are not currently capable of navigating using vision, even in clear weather conditions. Thus, a valuable source of navigation capability and backup instrumentation is currently not being utilized. This is of particular concern in the context of security and reliability, when external navigational aids may not always be available or reliable [4].

Navigating by sight is a complex task that requires the ability to recognize and contextualize terrain features from images under a wide variety of conditions. A recent review of the field by Couturier and Akhloufi [8] divides current approaches into two broad categories: Relative Visual Localization (RVL or frame to frame localization), which aims to update location by calculating movement when comparing one image frame to the next, and Absolute Visual Localization (AVL or frame to reference localization), which aims to determine location by comparing UAV/aircraft imagery to trusted georeferenced imagery, commonly from high-altitude survey flights or satellites. Of the two approaches, AVL has the notable advantage of being free from drift while RVL approaches tend to be more accurate than AVL approaches over short distances. Over longer distances, errors compound, as each error in estimating movement from the last frame adds to the error in estimating current location. AVL, however, can provide estimates of location that are independent of prior estimates. AVL, then, can be used as a complement to RVL. RVL provides high accuracy over short distances, while AVL provides periodic, independent estimates of location to prevent drift. This is similar to how GPS and IMUs work together, where IMUs provide high accuracy over short distances, while GPS provides periodic independent location information to prevent drift. Preventing drift is particularly important for long duration flights over large distances, typically made at high altitude. However, most prior work on AVL has focused on lower altitude flights by small UAVs, highlighting a need for investigation of AVL using higher altitude imagery.

Since AVL involves image processing, it is natural to investigate how the current state of the art in ground level image processing techniques performs when applied to this task. One of the most successful modern techniques for complex visual processing tasks is the Convolutional Neural Network (CNN). CNNs are very effective at learning complex relationships between input images and outputs. Furthermore, although the computational and memory resources needed to train a CNN are high, a fully trained model can be deployed at a considerably reduced cost, which is a desirable feature for UAVs whose computational resources may be limited.

Many frame-to-reference AVL approaches require the aerial platform to locally store reference imagery which must be retrieved during localization, meaning that storage and computational costs rapidly increase with the area of flight operation. Additionally, such AVL approaches that seek to increase accuracy/robustness by comparing to multiple reference data sets increase their storage and computational costs proportionally. In contrast, a pure neural network approach only requires the storage of the model parameters, and does not need to scale with the number of data sets used during training, and does not directly scale with the size of the ROI (larger ROIs will require larger networks with more parameters to encode information about locations in the ROI. It would take more parameters to encode a state than a county. However, because neural networks are known to be effective at efficiently encoding data (for example, in autoencoders), the size of the network needed to localize over a given ROI should not be directly proportional to the ROI's area). Finally, although neural networks have large requirements for data and computation during training, the speed of inference on a trained neural network is quick. This makes a pure neural network approach highly desirable if it can achieve useful accuracy over a large area. This paper aims to address the question of how effectively a modern CNN architecture that has been successfully used on ground-level imagery tasks can be repurposed for AVL.

When making neural networks for tasks involving recognizing objects at the ground level, there exist many high-quality pretrained models that can be used as a starting point. Unfortunately, there do not currently exist large, pretrained deep learning models for aerial imagery feature detection. In contrast to most prior work in the field (see Related Work section for details), we fully train our own model. This requires a large and varied data set.

Although large quantities of location-tagged aerial imagery exist, organizing and processing them for use in training a neural network remains an ongoing challenge. For this study, publicly available high-altitude images were used. Images are precropped and downscaled to allow for input into a CNN. Additional random crops augment the training data. One of the challenges of AVL is that the photographic properties of the input may vary considerably due to factors such as time of day, season, year, weather, and the properties of the camera used. This study therefore trains and tests on different image sources that vary by as many of these factors as possible.

Several configurations of CNNs were trained from scratch to regress latitude and longitude coordinates from high-altitude photographs (covering 1.5×1.5 km each) over the region of interest (ROI, the selected area within which localization was assessed), a ~ 32.2 by 32.2 km section of Washington County encompassing urban, suburban, and rural areas. The best performing architecture, Xception [6], was investigated further. An investigation into the use of probability distribution parameters as outputs in order to quantify model uncertainty was performed. The final models were trained on 12 data sets and tested on a 13th, with a separate model being trained with each data set acting as a holdout set to act as cross-validation. Performance was robust across all cross-validation training runs, and performance was high relative to the area of the ROI and size of the input images, where average errors were as low as .004% of the ROI's area and 8% of input image width, respectively. Future directions to improve CNN performance on this task, and better incorporate them into overall UAV navigation models, are discussed.

Related Works

Visual localization can be split into two broad categories: relative visual localization (RVL) or frame-to-frame localization, and absolute visual localization (AVL) or frame-to-reference localization. Although this paper is addressing the problem of AVL, RVL is the more mature field and the field of AVL has developed primarily out of a need to address drift in RVL approaches. Because RVL provides a location relative to the last location estimate, error inevitably compounds no matter how accurate the technique (unless the error is zero—a very high bar to cross!). Thus, even the most accurate RVL techniques have unbounded

error over long enough timescales. In application, AVL is best used together with RVL, with RVL providing high accuracy over short distances and AVL preventing drift from becoming unbounded. Additionally, RVL can only provide relative position updates - it cannot localize when there is no initial estimate of position, while AVL can.

The most straightforward type of RVL is visual odometry (VO) [19]. VO compares the current and prior observation, compares them, and then determines the movement of the platform based on differences. VO approaches evolved out of feature-based detection algorithms such as SIFT [16] (Scale-invariant Feature Transform), where certain features, crafted to be robust to image transformations caused by camera pose, are extracted from each image and then matched across images. The difference in the locations and orientations of these features within each image can then be used to estimate camera movement. Feature extraction and matching is computationally intensive, leading to interest in VO methods that do not use feature extraction (see [10] for an example). Additionally, some approaches now also incorporate additional information from IMUs to increase accuracy, an approach called Visual Inertial Odometry [15].

Because VO updates position from prior estimates, it is susceptible to drift over time. If the aircraft loops over prior locations (loop closure), then if the previously visited location is recognized the aircraft can reconcile its current and past location estimates, eliminating drift. This is approach taken in Simultaneous-Localization And Matching (SLAM) [9], where a map of prior visited locations is constructed simultaneously with location estimation. However, in the absence of loop closures (such as in straight-line flight), or if it fails to recognize a prior location, SLAM is still susceptible to drift. Additionally, SLAM requires additional computational resources to store and query its map of prior locations - and this cost increases over operation time as the number of prior locations grows. Thus, although effective, SLAM does not solve the problem of location drift in all situations, highlighting the need for AVL approaches. In contrast to RVL, which updates position based on a prior estimate, AVL compares the operational imagery during flight to a trusted reference to produce a location estimate. Since each comparison to the reference is made independently, each location estimate is independent of previous estimates and is not susceptible to compounding drift.

Couturier and Akhloufi recently published an excellent review of AVL [8] which covers

much of the recent publications in the field. From this review, it is seen that the most popular and currently most successful AVL methods use template or feature points matching, with only a minority opting to use deep learning techniques, and even then often only using deep learning techniques as a supplement to another matching technique. This is despite deep learning being the leading method in various ground-based tasks, including analogous problems such as self-driving cars. Neural networks are also being successfully applied to the related problem of cross-view geolocation, where ground level pictures of locations must be matched to aerial pictures of the same location (see for example Hu et al. [14]).

There are two major difficulties facing researchers attempting to leverage deep learning on aerial photographs: lack of existing models, and difficulty in obtaining sufficient train/test data. Although it is still possible to use the early layers of models trained on ground level imagery tasks, it is likely that the filters learned when training on ground level imagery are inefficient when applied to aerial imagery, especially high-altitude imagery, due to the vast difference in image scale and perspective. Nonetheless, because of the difficulties in training a deep model from scratch, most papers using them to-date have opted to fine-tune existing pre-trained ground-level models [see, e.g., [7, 1, 18, 11]]. Unlike our approach, none of these works attempt to directly determine location using a deep model, but instead use neural networks as steps along a template or segment matching pipeline. Thus, the potential of a purely CNN based AVL procedure remains unknown.

Additionally, because of the large computational cost of matching over a large area, many models employ some level of search space reduction such as a sliding window [11] or constant registration of the current position [18]. This presents a problem if the error in the estimate of position ever becomes too large: the reduced search space may no longer include the true position. In other words, these systems can suppress small-scale drift, but are still susceptible to drift resulting from larger errors. They also cannot localize without some estimate of initial position. Thus, they are susceptible to outlier estimates, system memory failure, and are unsuitable for recovering from an extended navigation blackout (e.g., flying through an extended fogbank or cloud in a GPS-denied environment).

Schleiss et al. [21] fully train a generative adversarial network, but use this to convert input images into a semantically segmented (to roads, buildings, and none) map-like image

to template match to an also segmented reference map (in this case, from OpenStreetMap, OSM). They do not employ any search space reduction, but their test area is only 560 m long and 680 m wide. Because their template matching technique uses a sliding window approach, it would likely require some form of search space reduction to remain computationally feasible over a larger area. Notably, they train and test on different (though geographically nearby) regions. The ability for this technique to generalize outside of the trained area is a considerable benefit, as it means that it is not necessary to train a model over the operational area specifically. However, the fact that the model must segment the image to match to a specific reference map creates two possible issues. First, as demonstrated in the paper, the model cannot differentiate areas with limited texture of the selected semantic categories (for example, a forest with no structures or roads). Second, although one can apply this technique to regions not trained on, the technique does require an accurate reference map segmented in the same manner as the model was trained on. In this case, the model could only be applied to where OSM has an accurate segmentation of the local terrain already available.

To our knowledge, only one other published work on AVL to-date has trained a deep CNN from scratch for direct regression AVL. Marcu et al. [17] trained a multi-stage multi-task architecture for simultaneous geolocalization and semantic segmentation. Their best approach uses a model that treats AVL as an image segmentation problem, and combines this with a simultaneous semantic segmentation that is then used to fine tune the image correspondence by matching identified roads. They also train one branch of their network for direct regression of latitude and longitude. However, the output of the direct regression branch is not the focus of their paper, and results are only reported in a single histogram (thus average error is not known). Additionally, although different train and test images are used, they appear to be from a common source and therefore have similar photographic qualities. Learning to identify locations across multiple photographic sources which may differ in time of day, season, year, and photographic platform is a considerably more difficult challenge.

3.2 Methods and Results

3.2.1 Data Choice and Processing

An effective deep learning approach to AVL should be robust to factors such as time of day, season, year, altitude, weather, angle of approach, and the camera used, maintaining accuracy under as many different conditions as possible. Producing robustness to these factors in a neural network requires large, high quality input data that includes the full range of plausible scenarios within it. Furthermore, all of these data must be georeferenced accurate to the desired scale. Unfortunately, such data is not readily available to the public in an organized form, which has hampered research efforts in this field.

Georeferenced imagery from satellite and high-altitude survey flights, however, are readily available, and serve as a reasonable proxy to high-altitude UAV images (indeed, an increasing number of such image sets *are* from high-altitude UAVs). In total, 13 data sets over Washington County, Arkansas were acquired (see Appendix 3.5 for detailed information on data sets and links to source data). These data sets differ in year taken (from 2006 to 2020), season, time of day, and camera used. All image sets used are from clear weather, daytime flights due to the vast majority of georeferenced, publicly available data being of this type. The data sets are also orthorectified using a mixture of digital elevation models - some from national elevation datasets and others from elevation data produced from the aerial triangulation results. For high altitude flights this alteration is relatively minor as the photographs are already taken nearly vertical angles and the terrain effects in this area are minimal at the flown altitudes. See Figure 3.1 for examples of 3×3 km crops and how they vary between a few of the datasets.

Since the data sets did not all completely cover Washington County, a 34.2 by 34.2 km square patch included in all data sets was chosen as the ROI within which localization would be assessed. This area includes both the urban centers of Fayetteville and Springdale, as well as the suburban and rural areas around them. The presence of highly varied terrain and land use - including both developed and undeveloped areas - makes this an especially good test of our approach's robustness to terrain character.

From each data set, 2000 3×3 km random crops and their locations were taken

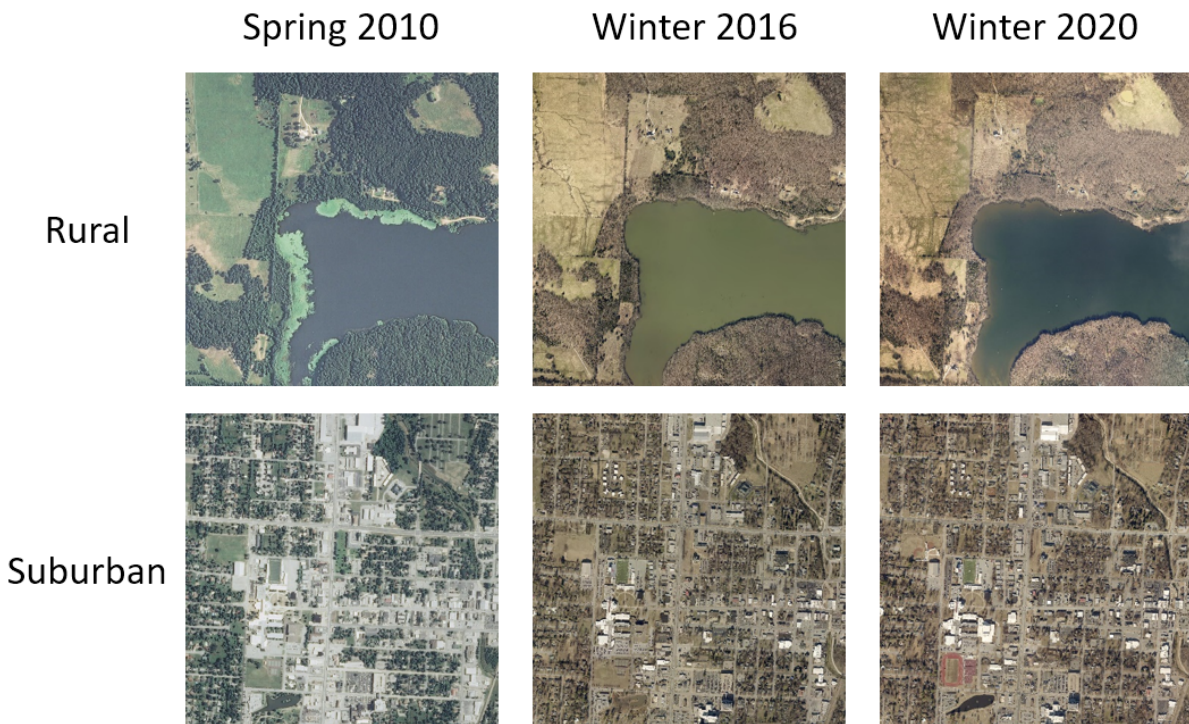


Figure 3.1: Example images from different data sets, illustrating the visual changes to two locations, one rural and one suburban, over different years and seasons.

from the ROI and downsampled to 1000×1000 pixels for speed of image loading and manipulation. During training of the neural network, each one of these crops is randomly cropped again to 1.5×1.5 km and downsampled to 224×224 pixels. Thus, a full epoch of training exposes the network to 2000 images from each data set, each of which is at least slightly novel due to random cropping. The labels for the network are the latitude and longitude of the image centers (in implementation the upper left-hand corner location was used as the label, as this is the native format of world files. Because all images are North–South aligned, the image center is always 750 m down and to the right of the upper-left hand corner, and so determining the location of the corner also determines the location of the image center), with locations adjusted as needed during cropping, scaled first to a range between 0 and 1 using a min-max scaler. The scaling applied is such that, over the entire dataset, the minimum latitude and longitude is 0 and the maximum is 1. Since the coordinate system used during processing increases in the west–east, south–north directions, the bottom left (southwest) corner of the ROI has coordinate (0,0), and the top-right (northeast) corner has coordinate (1,1) (thus the ROI is normalized to the unit square). When adjusting labels during random crops, the flat world assumption is used, which is justified because the images are high-altitude and orthorectified vertical. This also justifies the conflation of image centers and UAV location; for a camera looking straight down they will be approximately the same.

See Figure 3.2 for a visual overview of the image processing pipeline.

The offline initial crop and downsampling is done using bicubic interpolation, however the online resizing for loading into the neural net is done using bilinear interpolation to reduce computational time. The image crops are saved as jpegs to reduce file size. Although both resampling and jpeg compression can introduce artifacts in images, both training and test images go through the same downsampling process, so these artifacts are consistent at train and test time. Pilot runs (data not shown) showed no significant difference in final model accuracy from using different common resizing methods.

3.2.2 Network Architecture Selection

For initial selection of network architecture, we decided to test a variety of architectures already available in Keras[5] as built-in models (see Appendix /refappa for detailed information on implementation and hyperparameter choice. Model references: EffNet[24], ResNet [13], VGG [22], MobileNetv2 [20], Inception [23], Xception [6]). These include many of the most successful models for various classification data sets and competitions over the last decade. Because we were only interested in comparing the models, the data set used for classification was a limited one where training and test was restricted to only one data set (ADOP2017). Furthermore, fine tuning for each architecture was minimal, so this comparison should not be taken as a final verdict on model performance. Performance is reported in mean absolute error on the x, y in the converted coordinate scheme as this test was solely intended for a preliminary model comparison. Models were given 500 epochs to train. See Appendix 3.5 for further implementation details.

Results are shown in Table 3.1. Based on Xception’s high performance and relatively small model size, it was selected as the model of choice for further experiments. For subsequent experiments, Xception was modified to include additional fully connected and dropout layers before the output layer in order to provide additional regularization and capacity (these modifications are similar to some of the configurations used in [6]).

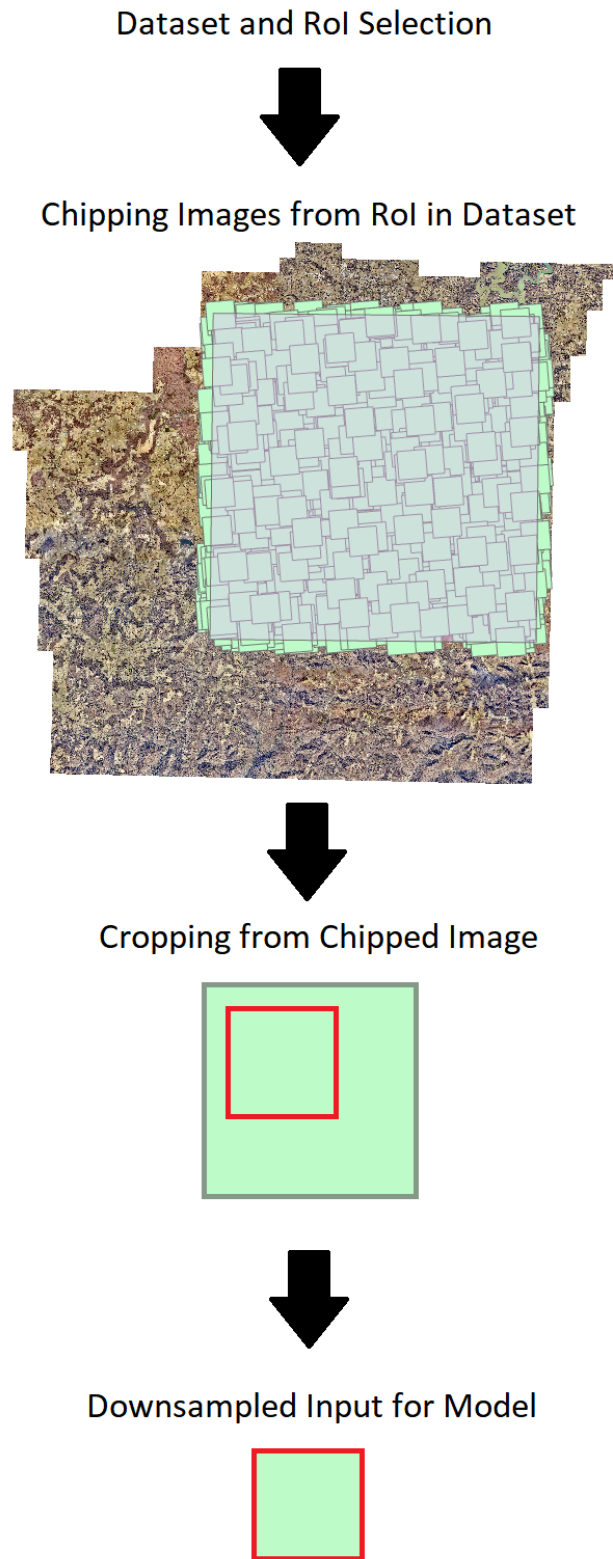


Figure 3.2: Figure demonstrating cropping setup. The transparent square over the county represents the ROI. The green squares are individual 3km x 3km image chips extracted from the base data set. 2,000 random chips are taken from each data set to ensure overcoverage, with different random chips for each data set. The final step before use in network training is a random crop from each chip, shown in red. Downsampling occurs at both the chipping and cropping stages. Chipping is an offline step performed once. Cropping is an online step performed during every epoch with different random crops.

Model	NMAE
EffNetB4	.25
EffNetB7*	.25
EffNetB0	.18
Resnet152V2	.09
VGG16	.08
VGG19	.08
VGG16Conv**	.07
Resnet152V2	.07
MobileNetv2	.06
Inception	.05
Xception	.04

Table 3.1: Model comparison for various built-in Keras models. Error is given as mean absolute error over the test set in normalized coordinates (NMAE). Note that performance should be about .25, the average distance to the center along either axis, if the model solely guesses (.5,.5), the middle of the ROI, as an output. *EffNetB7 became overfitted extremely early in training. **VGG16Conv was a custom modification of a smaller VGG-type architecture where pooling layers were replaced with strided convolution layers.

3.2.3 Field of View Comparison

As the altitude of an aerial platform increases, the field of view, and therefore the number of features that can be seen and used to navigate, increases. However, distance also decreases the ability to resolve smaller objects, and fine features are lost. In order to investigate this tradeoff within a CNN context, different field of views for the input images were tested while holding the input size fixed at 224×224 pixels. At the time this experiment was performed, the data set only had 11 of the 13 data sets in it. Holdout data set was held constant as the ADOP2006 data set for comparison. Training was performed for 250 epochs.

See results in Table 3.2. Results are reported in normalized RMSE (NRMSE) for simple comparison. Within the range studied, it was found that increasing field of view increased performance. There is a confounding factor that due to the processing pipeline, very large fields of view had fewer unique crops (that is, cropping a greater percentage of the source image reduces the difference between crops), however, this would be expected to decrease performance due to producing less varied training data.

3.2.4 Loss Function Comparison

For regression tasks there are numerous loss functions available to train a neural network. Since backpropagation training of neural networks is not a straightforward optimization relative to the loss function, it is not the case that using the desired metric as a

Input Width/Height	NRMSE
.25km	.3013
.5km	.2770
1km	.0671
1.5km	.0269
2km	.0171
3km	.0165

Table 3.2: Field of view comparison. Absolute accuracy improved as field of view increased despite loss in visual resolution and confounding factors that favored smaller fields of view. Error is given as RMSE in normalized coordinates (NRMSE).

loss function will lead to the best results as measured by that metric. That is, if our desired metric is to minimize the Euclidean distance between the predicted and actual location, the loss function that produces the best model may not be the Euclidean distance itself. It is therefore necessary to investigate loss functions and empirically observe their impact on model training.

For this experiment, three loss functions were investigated. First, mean squared error, which is the most common loss function used for regression tasks. Secondly, Euclidean distance (typically, Euclidean distance is used only as a metric, however it can be used as a loss function. RMSE and Euclidean distance are of the same degree with regards to the error, and differ only by the averaging operation in RMSE. When the number of dimensions and batch size are fixed, this difference will be a constant multiplicative factor. That is, RMSE and Euclidean distance produce the same pattern of losses differing only in scale, which can be changed by altering the learning rate), which is our desired metric to minimize. Thirdly, the output layers are changed to be the mean and standard deviation of a normal distribution, and the loss function is the negative log-likelihood. This would allow the model to output a quantity related to its confidence in the prediction at a small computational cost.

Each model was trained for 500 epochs. Because of the difference in gradients due to choice of loss function, learning rate was individually tuned for each model based on pilot runs (data not shown). See Appendix 3.5 for implementation details.

Since the distribution-output model trained with negative log-likelihood trained more quickly, had a higher accuracy, and gave additional information in the form of the output distribution’s standard deviation, it was selected as the model of choice for subsequent ex-

Loss Function	Normalized Mean Euclidean Error
NegLogLikelihood	.0041
MSE*	.0198
Euclidean Distance*	.0232

Table 3.3: Loss function comparison. Holdout set was help constant as AO15. Error is the average distance error over the entire holdout set in normalized distances.

AO20 Run	NRMSE
1	.0556
2	.0474
3	.04811
4	.0530
5	.0541

Table 3.4: Run replicability experiment results. Results are given in RMSE of the normalized error distances (NRMSE).

periments.

The results are shown in Table 3.3.

3.2.5 Training Replicability

To help estimate the effects of stochasticity during training on the final accuracy, five runs of the exact same model were trained for the same number of epochs and their accuracies compared. See Table 3.4. Results showed reasonably low inter-run variation, which suggests that training results are replicable.

3.2.6 AVL Results

The Xception-based model was trained on 12 data sets and tested on one holdout dataset. Each dataset was used as a holdout set in a separate training run to cross-validate the results. Models were trained with both independent normal and bivariate normal distributions as the final layer, with the negative log-likelihood used as training loss. On the validation step, since it was noticed that the random cropping in the data pipeline would lead to occasional sampling outside of the ROI and to undersampling of edge regions during training, the ROI was shrunk slightly by $\sim 5\%$ on each side. Since the input images are 1.5 km a side, this shrinks the ROI to about 32.2 km a side or $\sim 1036 \text{ km}^2$. The second cropping step during validation was always taken from the upper-left corner of the input image so that the validation input images were the same for direct comparison.

Test Set	Average Error	
	Independent Normal	Bivariate Normal
ADOP2006	188.2m	214.4m
ADOP2017	172.7m	206.8m
NAIP2006	189.2m	215.9m
NAIP2009	323.8m	320.2m
NAIP2010	146.4m	160.2m
NAIP2013	129.9m	164.8m
NAIP2015	173.4m	185.0m
CAST2008	183.4m	251.3m
AO15	124.3m	168.9m
AO16	115.5m	156.5m
AO17	127.6m	169.3m
AO19	141.6m	174.1m
AO20	148.7m	166.4m

Table 3.5: Xception results on different holdout sets. Models were trained for 500 epochs. Error is reported in Euclidean distance.

See Table 3.5 for results. See Figure 3.3 for a visual representation of error scale relative to ROI for one representative model. Results were small relative to the ROI and the input’s geographic size of 1.5×1.5 km. Results were largely robust to choice of holdout set. The larger error of the NAIP2009 holdout set was driven by cloud cover in part of the data set.

3.2.7 Uncertainty Calibration

One of the advantages of using a distribution output layer and a likelihood based log function is that the network produces an additional output related to prediction confidence: the standard error of the output normal. We did find that the standard deviation was correlated with error size for predictions. However, as model accuracy increased, standard deviations did not shrink at the same rate as the error sizes, leading to underconfidence, see Figure 3.4. A similar pattern of underconfidence was observed for both bivariate (bivariate confidence intervals were calculated jointly, using a Chi-square distribution, whereas the independent normal CIs were calculated independently for X and Y axes) and independent normal runs.

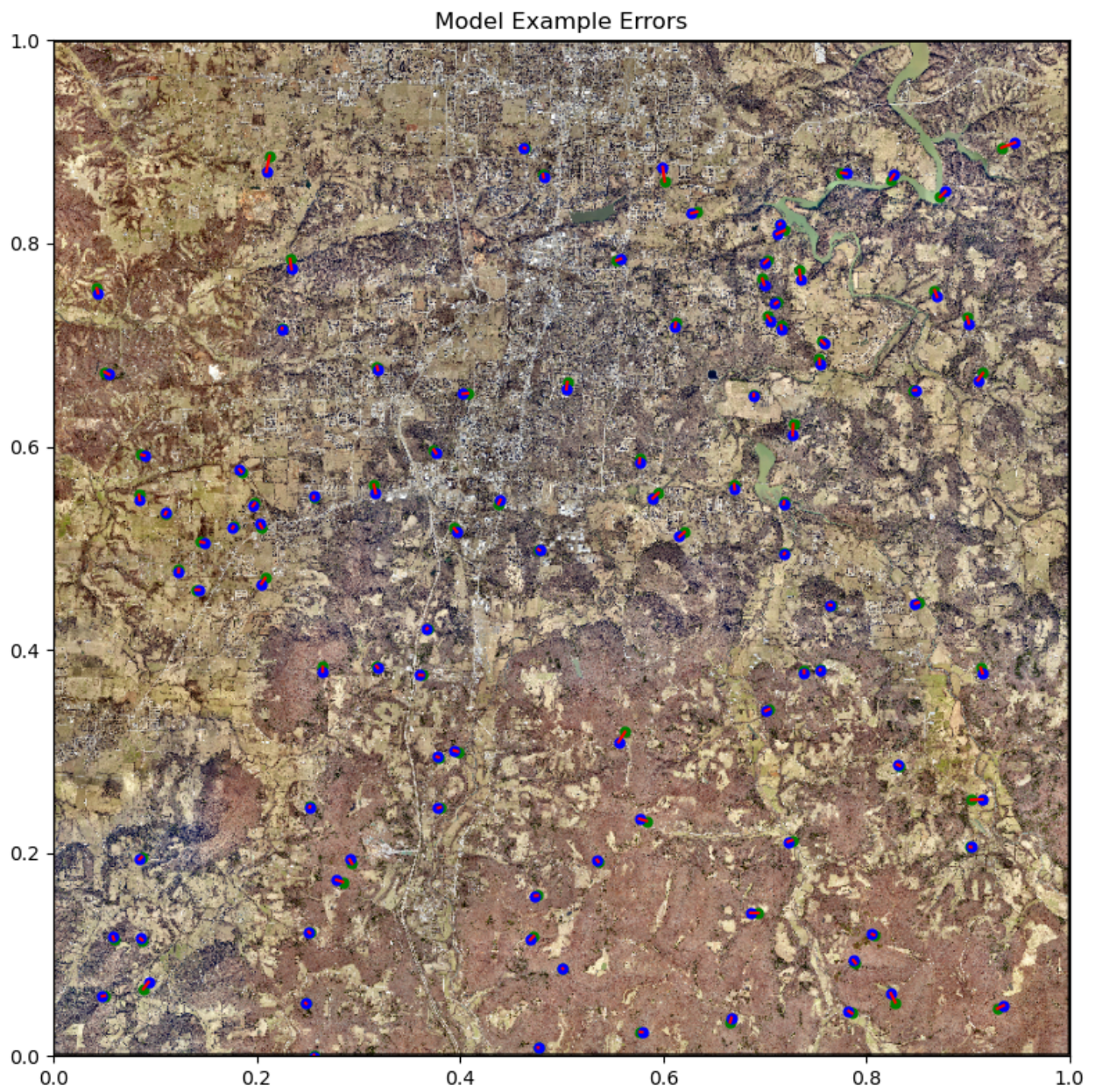
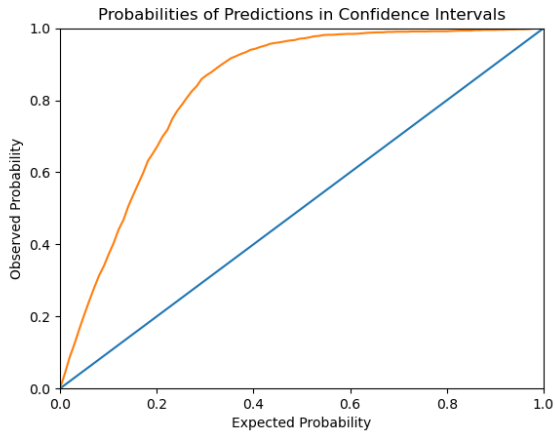
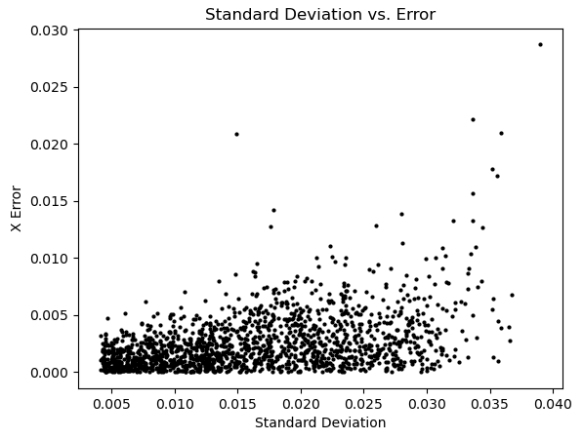


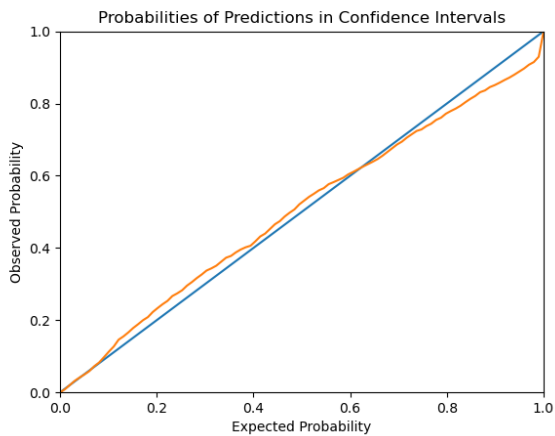
Figure 3.3: Figure demonstrating 100 sample errors of one of the runs (WaCo2020). Green is the true location, blue is the prediction, where predictions are connected to their true location by a red line. The background is of the holdout data set over the ROI.



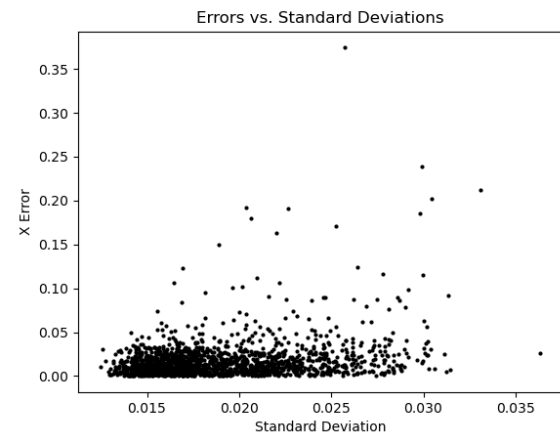
(a) Calibration of Overconfident, Accurate Model



(b) Scatter Plot of Errors, Overconfident Model



(c) Calibration of Less Accurate Model



(d) Scatter Plot of Errors, Inaccurate Model

Figure 3.4: Comparison of uncertainty calibration between an accurate model and a less accurate model. The units in b and d are normalized over the unit square ROI. The first model, shown in parts a and b, had an average error of $\sim 149\text{m}$. It has poor uncertainty calibration; more of the true centers are contained in a given percent confidence interval than expected, meaning the model is underconfident. The second model, shown in c and d, was incompletely trained, and only achieved an average error of 965m . However, it showed better uncertainty calibration. Note, however, that some of the error sizes are unrealistically large given the standard errors, indicating the true uncertainty distribution is not Gaussian.

3.3 Discussion

3.3.1 Accuracy and Loss Function Choice

This paper sought to investigate how suitable CNNs that have been successful on ground level tasks are for the purpose of aerial AVL. Results were highly encouraging. With limited tuning, and using only 12 training data sets, we were able to train Xception to successfully localize over a $>1000\text{ km}^2$ geographic area. Our best model was able to localize with an average error as small as 115.5m on $1.5 \times 1.5\text{km}$ input images. Although the absolute size of this error is larger than many methods previously reported, the error relative to the input size of the image and the ROI is small. Moreover, we were able to include an additional measure of prediction uncertainty in the form of the output distribution's standard error. Not only did this inclusion not reduce performance, switching to a distribution output with

negative log likelihood loss actually increased accuracy while decreasing training time.

This was a surprising result, as we expected the increase in output complexity to decrease accuracy. We hypothesize that the decreased training time and increased accuracy may be due to the sharper gradient increase with error distance attributable to the likelihood loss model. That is, because likelihood scales sharply to zero with error size when using a normal distribution as the output distribution, the negative log-likelihood gradient scales enormously with error size, far more so than for MSE where it only grows quadratically. Overfitting also was observed to be less severe in the probabilistic models. Because of the potential for large gradients, it was necessary to cap gradient size whenever we trained the distribution output models, or it was extremely likely to produce NaN loss at some point due to overflow. This was especially important in bivariate models. The bivariate models, while theoretically more expressive than the models with two independent normals, were slightly less accurate overall, indicating that the gain in model expressivity was not worth the increase in model complexity. Future work may wish to investigate choice of output distribution further. Multi-modal distributions, as in mixture density networks, may be able to better encapsulate uncertainty caused by distant locations being visually similar.

Results were robust to choice of holdout set, with the caveats that the error was slightly lower for the grouping of AO datasets, likely due to the larger number of similar datasets in the training set, and a slightly worse accuracy when using the NAIP2009 dataset. Upon further investigation, the source of this increased error was traced to a cloud covered region in the NAIP2009 dataset where the model performed poorly, as it was not trained on any cloudy data sets. In future studies, it would be beneficial to try and train over a variety of weather conditions if appropriate data sets can be found or artificially generated (e.g., by randomly adding various levels of cloud cover). Another approach would be to mask clouds in the training data as a preprocessing step.

Differences in choice of image and region domain make direct comparisons between different AVL methods difficult. The absolute error of our is large compared to those reported by most prior works, while the relative error compared to the geographic size of the ROI and the input images is small. That is to say, an error of 50 m while operating at low altitudes may mean that the location estimation is entirely outside of the image field of view, and

the UAV is in serious danger of becoming lost. On the other hand, an error of 50 m while operating at a height of several km may mean that the estimated location was only a few pixels away from the true center of the input image.

However, this paper is not seeking to promote this particular model as a ready-to-implement solution, but rather to see whether a very simple approach using solely CNNs and image inputs could be successful on this task. Undoubtedly, with additional tuning and training examples, the current results could be significantly improved upon. Moreover, overfitting was a large issue during training and had to be aggressively controlled with dropout and L1/L2 regularization methods. This is encouraging because it suggests that the models are not utilizing their full expressive capability. This suggests that models of Xception's size are capable of performing AVL over a considerably larger geographic area than the ROI here. Or, alternatively, a considerably smaller model could be used without greatly affecting accuracy, saving on computation.

3.3.2 Uncertainty Calibration

The standard deviations produced by the model do correlate with error size. However, they are too conservative. This miscalibration tended to increase with model accuracy. Uncertainty miscalibration increasing with accuracy is a known problem with deep networks[12]. However, in classification problems models have tended to become overconfident at high accuracy, whereas here the model becomes underconfident. This may be due to the choice of a normal distribution to model the uncertainty. Since you can have visually similar regions that are geographically distant, the true uncertainty distribution should be multi-modal.

The Gaussian distribution has very thin tails that cause the likelihood of outliers to fall rapidly. In this case, if the true uncertainty has more probability mass at long distances from the mean than a Gaussian distribution can represent (which seems likely), then we should expect the standard deviations to be conservative. This is because the occurrence of occasionally large errors due to multi-modality will drive the standard deviation estimate up substantially. This explanation would also explain why the underconfidence increases with training. Notice that even in the undertrained, better-calibrated model, the error vs. SE plot shown in Figure 3.4d shows some $>10\sigma$ events - which would be virtually impossible if the

error landscape was truly Gaussian and the model was correctly calibrated. The calibration of model uncertainty remains an area for future investigation.

3.3.3 Computational Considerations

A large issue for many current AVL techniques is computational load. In this domain, we believe deep learning offers many advantages. Although deep models take considerable computational resources to train, when put into production where they are only doing forward passes on single samples they are efficient. The Xception architecture is on the smaller end of famous architectures, weighing in at about 88 MB compared to VGG16’s 528 MB (from <https://keras.io/api/applications/>, accessed on Oct 7, 2021 This is parameter storage size, not required memory for deployment, and does not include the fully connected layers we included in our Xception model.). On a device with a GPU, which can easily be fit into larger drones, a forward pass of Xception takes under 10 milliseconds [2]. Even without a GPU, a single forward pass of Xception is only on the order of ~ 1 s or less, depending on the CPU used (from personal experience), which is still feasible for navigation use at higher altitudes where views change more slowly.

As GPUs and TPUs are increasingly being incorporated into mobile devices, even smaller drones would be able to take advantage of this speed. Xception only requires 1–2 GB of memory [2] for inference, making it it plausible to run it on mobile devices. A larger savings comes from system memory and storage; rather than having to store an entire database of template images to match to, which may also need to be partially loaded into memory, the drone would only have to store a single trained model. Furthermore, in a production setting, techniques are available to compress/prune trained networks to greatly reduce their size while minimally impacting their performance, yielding further speed and memory improvements (see, e.g., [3] for a recent overview of pruning techniques).

3.3.4 Input and Output Choices

The current CNN only addresses obtaining horizontal coordinates from nadir images. The reason we chose to focus on this is because we feel this is the core challenge of AVL. Full six degree-of-freedom pose estimation is undoubtedly extremely important. However,

four of those parameters (roll, pitch, and yaw angles and flying height) can be obtained from instruments directly without reference to any external systems. While it would certainly be beneficial to also have an estimate of pitch, yaw, roll, and altitude from vision, it is not as pressing since internal systems are much less susceptible to interference. You cannot, for instance, spoof Earth’s gravity to deceive an accelerometer. For this reason, we focus solely on the challenge of horizontal positioning. This is the heart of visual localization, and requires more than creating better, more reliable sensors.

Similarly, we did not concern ourselves here with input images of different scales or rotational orientation. In an applied setting, this would need to be dealt with, either by training on such data, or more simply by preprocessing image inputs to the appropriate orientation and scale, which can be done if we assume that the UAV has at least a rough idea of its orientation, altitude, and camera position. Fortunately, as discussed above, these can all be tracked solely by internal systems such as a compass or altimeter, which are cheaply available and nonreliant on external systems. Since this preprocessing can be performed after downsampling, its computational cost would be small. It is worth noting that, although all data sets were downsampled to the same size and covered the same geographic extent, their differing initial ground sample distance (GSD) would still have an effect on the final input images. However, the model proved capable of generalizing across data sets with different initial GSDs.

A more complicated issue is that of image obliqueness. Especially at low altitudes, image features will look very different based on the angle of the object to the camera. We kept the problem as simple as possible, opting to use high-altitude orthorectified vertical imagery on a projected surface. Such imagery is also far more widely available than low-altitude imagery. For images that are not extremely oblique, orientation angle can be used to convert images to their nadir view with only some non-correctable terrain distortions (e.g., minor occlusion) remaining, standardizing the input. Addressing the additional challenges of extremely oblique input imagery that cannot be converted to nadir view easily remains for future work.

The height experiment presented here suggests that the problem becomes more difficult as geographic field-of-view decreases, even though resolution increases. This is expected

as a larger field of view means more opportunities to include relevant identifying features. However, the decrease in accuracy, especially at the 0.5 km and 0.25 km levels, is so extreme as to suggest other factors are also at work. Although the higher resolution of a smaller field-of-view might allow the discrimination of finer features, these smaller features are more likely to be transient (e.g., cars, shadows, foliage patterns) and therefore not robust. CNN architectures have a built-in sensitivity to features of a certain size due to their architectures. For lower altitude images, the most identifying features of a location may be quite large relative to the size of the image, and therefore difficult for CNNs optimized for detecting more local features to use. It is notable that the Xception and Inception architectures, which performed the best in the architecture comparison, are designed to include filters of multiple different sizes to detect features at varying scales.

A future experiment would be to compare increases in resolution while holding geographic field-of-view constant. However, it is difficult to compare this in a principled manner since model size increases with input size unless modifications are made, but such modifications also have implications on the model’s performance. The fact that increasing field-of-view increases AVL performance suggests that, in models capable of handling multiple scale inputs, a navigation strategy would be to raise altitude and increase field-of-view as much as possible. This is a strategy that humans also use.

3.3.5 Future Directions for AVL Deep Models

The current results are encouraging but also highlight the need for the development of architectures specifically suited for the problem domain. Many aspects of architectures built for ground-level classification tasks are ill-suited for AVL. A pressing example is that most such architectures specifically aim for a high level of translational invariance in their results. This is because in ground-level classification tasks the specific location of objects in location is often irrelevant (a dog remains a dog where ever in the image it is). In AVL, however, the location of features in the image is critical to precise localization. Pooling layers, which reduce the resolution of input, are known to promote translational invariance.

To illustrate the issue, consider that the Xception model has four max-pooling layers, each of which approximately halves the spatial resolution. Thus, we can roughly estimate

that the spatial resolution of the model before the dense layer may be on the order of 1/16th the input resolution. Given the input resolution of 224×224 px for a 1.5×1.5 km, this would suggest a final spatial resolution on the order of 100 m for the Xception model, which our best models approach in accuracy. Although this should not be taken as a hard limit on the model's possible accuracy, since information can combine across filters and areas, it does suggest that improving accuracy beyond this point will be more difficult for the model. Thus, reducing the level of pooling, or otherwise increasing spatial resolution, may lead to improved model performance on AVL. However, pooling layers also play an important role in reducing model size to manageable levels, and regularizing the network against overfitting on fine features that are unlikely to be robust. Model size also increase quickly with input resolution. The problem of designing architectures optimized for AVL is not a simple one, and requires additional investigation.

Secondly, a major weakness of the CNN-only method is the necessity of training directly over the area of operation. However, this is a weakness shared by many other proposed AVL techniques. Even those techniques which are based on template-matching must always have templates of their full area of operation loaded into memory in order to match. Indeed, the CNN only method can be conceptualized as an implicit template-matching method. It is matching against representations of locations contained implicitly within the model's weights. When thought of in this way, the CNN training process is a way of efficiently encoding all location templates so that they can all be matched against quickly every time an image is presented. Notably, this approach does not require an increase in model size to incorporate additional datasets into the training process.

The issue of needing to gather data and devote computational resources to train a model for each new area of operation, however, remains a serious issue. This highlights the need for established, quality models specifically designed for AVL. If such models existed for any region, then the early layers could be used pre-trained. Adapting the model for a new region would be then be a fine-tuning task. This would greatly reduce the training time and amount of data needed for use in new areas. One of the most pressing needs right now is an established, standardized, high-quality large data set to use as a benchmark for AVL. Because publicly available image sets favor good visual conditions, finding data sets for poor

visual conditions such as inclement weather or at night is especially challenging.

Finally, most prior work using deep models on this problem have combined them with other techniques. Although we do not do so here, we believe that this is a sound technique. Our approach of a pure CNN model has the advantage of not requiring any prior estimate of location to work. In an applied setting, we think that our approach would be most useful as a periodic check for drift on some other technique, or a way to initialize the position estimate. Currently, visual location techniques tend to be split into two broad categories: techniques that are precise over short distances but susceptible to drift (RVL), and techniques that are less precise but are not susceptible to drift (AVL). A hybrid approach that uses RVL over short distances while periodically checking and rebasing with AVL can combine these strengths, in a manner analogous to how IMUs and GPS work together synergistically.

3.4 Conclusions

A pure neural network approach was shown to be viable for AVL over a large geographic area. First, several architectures that had previously shown success at ground-based classification tasks were tested, with the Xception model showing the best performance. The Xception model was then modified to output parameters of a normal distribution describing the location estimate, training with the negative log-likelihood as the loss function. Subsequently, a modified Xception-based model achieved an average error of ~ 166.5 m across 13 different cross-validation training runs on different holdout sets. This is good accuracy compared to the $1.5 \text{ km} \times 1.5 \text{ km}$ size of the input images, and the ~ 32.2 by 32.2 km ROI. Additionally, modifying the model to output parameters of a normal probability distribution, and training with negative log-likelihood loss, not only did not decrease performance, but increased model accuracy. The standard deviations of the resulting distributions did correlate with error variance, showing potential usefulness as a measure of estimate uncertainty, although the fully trained models were underconfident.

The direct neural network approach presented here presents several attractive properties from a computational and performance standpoint. First, the storage requirements are minimal, requiring only the trained model parameters to be onboard the platform. Second, inference is fast and consistent (every forward pass requires the same number of operations,

there is never a possibility of needing to expand operations as with some other approaches that can fail to produce a match/estimate). Third, computational requirements during flight do not grow proportionally with the amount of datasets used for training, or with the area of the ROI being considered. This third property is especially important, as it means that this approach is potentially scaleable to much larger areas of operation with a proportionally minor increase in model size required.

In order to move from concept to application, more work has to be done on optimizing the neural network architecture, incorporating other techniques for fine tuning of position estimation, and, especially, the development of appropriate data sets for training and testing. This study shows that a neural network AVL models can be trained to be robust to time of day, season, camera characteristics, and even year that the data set was gathered. However, the ability of existing models to generalize across different camera angles, as well as to operate during night-time or during inclement weather, has yet to be demonstrated. To train models to attempt to solve these problems will take a substantial effort in data gathering and acquisition.

Author Contributions

Conceptualization, J.C. and C.R.; Investigation, W.H. and C.R.; Methodology, W.H., J.C. and C.R.; Software, W.H., Validation, W.H., J.C., and C.R.; Resources, J.C. and C.R., Data Curation, W.H.; Writing—Original Draft Preparation, W.H.; Writing—Review and Editing, W.H., J.C., and C.R.; Visualization, W.H.; Supervision and Project Administration, C.R. All authors have read and agreed to the published version of this manuscript.

Data Availability

Details on all data sets used in this study are available, with means of public access, in the Appendix..

Acknowledgments

The authors would like to acknowledge CAST, the Center for Advanced Spatial Technologies, the Arkansas DART (Data Analytics that are Robust and Trusted) project, and

the University of Arkansas, Fayetteville.

Abbreviations

The following abbreviations are used in this manuscript:

MDPI	Multidisciplinary Digital Publishing Institute
UAV	Unmanned Aerial Vehicle
IMU	Inertial Measurement Units
GNSS	Global Navigation Satellite System
GPS	Global Positioning System
CNN	Convolutional Neural Network
AVL	Absolute Visual Localization
RVL	Relative Visual Localization
OSM	OpenStreetMap
ROI	Region Of Interest
VO	Visual Odometry
SLAM	Simultaneous-Localization And Matching
GSD	Ground Sample Distance
RMSE	Root Mean Squared Error
NRMSE	Normalized Root Mean Squared Error
MAE	Mean Average Error
NMAE	Normalized Mean Average Error
GPU	Graphics Processing Unit
CPU	Central Processing Unit
TPU	Tensor Processing Unit

3.5 Appendix

3.5.1 Neural Network Implementation Details

Final AVL Runs

Both the bivariate and independent normal implementations were trained on an Xception architecture with no top layers as implemented in Keras in Tensorflow 2.4. On top of this implementation, we added global average pooling and two fully-connected layers with 4096 hidden units each, a configuration described in the Xception publication. Regularization was extremely important to prevent overfitting, so we added 3 dropout layers with dropout set to 0.5, one after the global average pooling layer and the other two after the fully connected layers.

All models were trained with the Adam optimizer with initial learning rate 1×10^{-4} , with a gradient clipvalue of 5. Because the probability mass of the normal distribution falls off very rapidly with distance from the mean, NLL gradients can become exceedingly large when the model makes a large error. Thus, for the negative log-likelihood models, gradient clipping is very important. The loss function was negative log-likelihood. The validation metric was RMSE on the holdout set. l1 and l2 regularization were set to 1×10^{-5} and 1×10^{-3} , respectively, and applied to all eligible layers. Learning rate was decreased by a factor of 0.7 every 50 epochs, and an additional 0.7 every 70 epochs without improvement on the validation set (this rarely occurred in practice). Models were trained for 600 epochs or 3 days on a single Nvidia V100 GPU (in practice, models typically achieved about 540 epochs). The best epoch performance on the validation set was reported. Models typically had plateaued but not reached full convergence by 540 epochs, but it was decided not to commit more computational resources for possibly marginal improvements. The main bottleneck was image loading/processing for input into the model, so this training time may be greatly improvable.

Other Runs

The loss function experiment settings and data set was identical to the AVL experiment, except that MSE and EDL had a learning rate of 1×10^{-3} , and of course had as their

output simple dense layers instead of dense layers that fed into distribution layers.

The height experiment was run on an earlier form of the final data set which was unbalanced in number of samples per input data set. Most data sets had 1250 image inputs, but a few had more. Settings were identical to the AVL run except for the following: the optimizer was stochastic gradient descent with 0.9 Nesterov momentum, the initial learning rate was 4×10^{-4} , except for the first 30 epochs burnin period where it was 1×10^{-4} to avoid early exploding gradients as at this time no gradient clipping was used. The learning rate decayed by 0.7 every 35 epochs without improvement in the validation set RMSE. The validation set was always ADOP2006.

The stochasticity experiment was performed with the same settings as the height experiment, with the following differences: initial learning rate after the first 30 epochs for the 2 km, 3 km, and 0.25 km experiment was 7×10^{-4} , and their burnin period was 20 epochs instead of 30. The other experiments in this series have a longer burnin and lower initial learning rate because they had to be restarted due to initial run attempts repeatedly ending in NaN training losses.

The architecture experiment had all architectures in their default configuration in tensorflow nightly version 2.3.0.dev20200611, `include_top = true`, except for VGG16Conv which differed from VGG16 by reducing depth and replacing max pooling layers with strided convolutions. The optimizer was stochastic gradient descent with 0.9 Nesterov momentum, initial lr rate 0.001, lr rate decay of 0.7 every 100 epochs, 500 epochs training time with best validation result taken as final result. As mentioned, this run was done on a benchmark data set where train and test data were both from ADOP2017 crops. In this data set, which was from a slightly differently positioned and larger square region of Washington County, there were 10,000 images. Train/test split was 80%/20%.

3.5.2 Data Set Characteristics

ADOP URLs: <https://gis.arkansas.gov/programs/arkansas-digital-ortho-program-adop/>, on Oct 7, 2021. <http://geostor-imagery.geostor.org.s3.amazonaws.com/index.html?prefix=State/ADOP/>, accessed on Oct 7, 2021. NOTE: 2001 ADOP is false color infrared! Orthophoto ground sample distance (GSD) is 1 m for 2006; 0.3 meters for

Test Set Name	Source	Foliage Season	Source GSD
ADOP2006	Arkansas Digital Ortho Program	Spring/Summer	1m
ADOP2017	Arkansas Digital Ortho Program	Winter/Fall	1m
NAIP2006	National Agriculture Imagery Program	Spring/Summer	2m
NAIP2009	National Agriculture Imagery Program	Spring/Summer	2m
NAIP2010	National Agriculture Imagery Program	Spring/Summer	2m
NAIP2013	National Agriculture Imagery Program	Spring/Summer	2m
NAIP2015	National Agriculture Imagery Program	Spring/Summer	2m
CAST2008	Center for Advanced Spatial Technologies	Mixed Foliage Summer/Fall	0.3m
AO15	Washington County Assessor's Office	Winter/Fall	.3m
AO16	Washington County Assessor's Office	Winter/Fall	0.15m/0.23m
AO17	Washington County Assessor's Office	Winter/Fall	0.23m
AO19	Washington County Assessor's Office	Winter/Fall	0.15m
AO20	Washington County Assessor's Office	Winter/Fall	0.15m

Table 3.6: Data set table.

2017 2006 is three-band true-color and three-band color-infrared, 2017 is true-color. NAD83 datum and UTM Zone 15 projection.

NAIP URLs: <https://www.fsa.usda.gov/programs-and-services/aerial-photography/imagery-programs/naip-imagery/>, accessed on Oct 7, 2021. <http://geostor-imagery.geostor.org.s3.amazonaws.com/index.html?prefix=State/USDA/>, accessed on Oct 7, 2021.

This data set contains imagery from the National Agriculture Imagery Program (NAIP). NAIP acquires digital ortho imagery during the agricultural growing seasons in the continental U.S. NAIP provides four main products: 1 m ground sample distance (GSD) ortho imagery rectified to a horizontal accuracy of within ± 5 m of reference digital ortho quarter quads (DOQQ's) from the National Digital Ortho Program (NDOP); 2 m GSD orthoimagery rectified to within ± 10 m of reference DOQQs; 1 m GSD ortho imagery rectified to within ± 6 meters to true ground; and, 2 m GSD ortho imagery rectified to within ± 10 m to true ground. The tiling format of NAIP imagery is based on a $3.75' \times 3.75'$ quarter quadrangle with a 300 m buffer on all four sides. NAIP quarter quads are formatted to the NAD83 datum and UTM Zone 15 projection.

CAST2008 URLs: Collected for the Washington County Assessors office by Pictometry (now EagleView). 0.30 cm GSD. This was an early collection that drove the standards for the later sets from AO. Collected July 2008. Available on request from the authors. NAD83 datum and UTM Zone 15 projection.

AO URLs: Imagery is publicly served as basemaps at: <https://arcserv.co.washington.ar.us/portal/apps/webappviewer/index.html?>, accessed on Oct 7, 2021. Inquire at Assessor’s Office about access to source rasters: <https://www.washingtoncountyar.gov/government/departments-a-e/assessor>, accessed on Oct 7, 2021.

Collected for the Washington County Assessors office by Pictometry (now EagleView). Has the following GSDs: 2015, 12in; 2016, 6/9in depending on area; 2017, 9in; 2019, 6in; 2020, 6in. NAD83 datum and UTM Zone 15 projection.

References

References

- [1] K. Amer, M. Samy, R. ElHakim, M. Shaker, and M. ElHelw. “Convolutional neural network-based deep urban signatures with application to drone localization”. In: *Proceedings of the IEEE International Conference on Computer Vision Workshops*. 2017, pp. 2138–2145.
- [2] S. Bianco, R. Cadene, L. Celona, and P. Napoletano. “Benchmark analysis of representative deep neural network architectures”. In: *IEEE Access* 6 (2018), pp. 64270–64277.
- [3] D. Blalock, J. J. G. Ortiz, J. Frankle, and J. Gutttag. “What is the state of neural network pruning?” In: *arXiv preprint arXiv:2003.03033* (2020).
- [4] C. Bonebrake and L. Ross O’Neil. “Attacks on GPS Time Reliability”. In: *IEEE Security Privacy* 12.3 (2014), pp. 82–84. DOI: 10.1109/MSP.2014.40.
- [5] F. Chollet et al. *Keras*. <https://keras.io>. 2015.
- [6] F. Chollet. “Xception: Deep Learning with Depthwise Separable Convolutions”. In: *CoRR* abs/1610.02357 (2016). arXiv: 1610.02357. URL: <http://arxiv.org/abs/1610.02357>.
- [7] D. Costea and M. Leordeanu. “Aerial image geolocalization from recognition and matching of roads and intersections”. In: *arXiv preprint arXiv:1605.08323* (2016).
- [8] A. Couturier and M. A. Akhloufi. “A review on absolute visual localization for UAV”. In: *Robotics and Autonomous Systems* 135 (2021), p. 103666.
- [9] M. Dissanayake, P. Newman, S. Clark, H. Durrant-Whyte, and M. Csorba. “A solution to the simultaneous localization and map building (SLAM) problem”. In: *IEEE Transactions on Robotics and Automation* 17.3 (2001), pp. 229–241. DOI: 10.1109/70.938381.
- [10] C. Forster, M. Pizzoli, and D. Scaramuzza. “SVO: Fast semi-direct monocular visual odometry”. In: *2014 IEEE International Conference on Robotics and Automation (ICRA)*. 2014, pp. 15–22. DOI: 10.1109/ICRA.2014.6906584.
- [11] H. Goforth and S. Lucey. “GPS-denied UAV localization using pre-existing satellite imagery”. In: *2019 International Conference on Robotics and Automation (ICRA)*. IEEE. 2019, pp. 2974–2980.

- [12] C. Guo, G. Pleiss, Y. Sun, and K. Q. Weinberger. “On calibration of modern neural networks”. In: *International Conference on Machine Learning*. PMLR. 2017, pp. 1321–1330.
- [13] K. He, X. Zhang, S. Ren, and J. Sun. *Identity Mappings in Deep Residual Networks*. 2016. arXiv: 1603.05027 [cs.CV].
- [14] S. Hu, M. Feng, R. M. H. Nguyen, and G. H. Lee. “CVM-Net: Cross-View Matching Network for Image-Based Ground-to-Aerial Geo-Localization”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2018.
- [15] S. Leutenegger, S. Lynen, M. Bosse, R. Siegwart, and P. Furgale. “Keyframe-based visual–inertial odometry using nonlinear optimization”. In: *The International Journal of Robotics Research* 34.3 (2015), pp. 314–334. DOI: 10.1177/0278364914554813.
- [16] D. G. Lowe. “Distinctive image features from scale-invariant keypoints”. In: *International journal of computer vision* 60.2 (2004), pp. 91–110.
- [17] A. Marcu, D. Costea, E. Slusanschi, and M. Leordeanu. “A multi-stage multi-task neural network for aerial scene interpretation and geolocalization”. In: *arXiv preprint arXiv:1804.01322* (2018).
- [18] A. Nassar, K. Amer, R. ElHakim, and M. ElHelw. “A deep cnn-based framework for enhanced aerial imagery registration with applications to uav geolocalization”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*. 2018, pp. 1513–1523.
- [19] D. Nister, O. Naroditsky, and J. Bergen. “Visual odometry”. In: *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2004. CVPR 2004*. Vol. 1. 2004, pp. I–I. DOI: 10.1109/CVPR.2004.1315094.
- [20] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen. *MobileNetV2: Inverted Residuals and Linear Bottlenecks*. 2019. arXiv: 1801.04381 [cs.CV].
- [21] M. Schleiss. “Translating aerial images into street-map-like representations for visual self-localization of UAVs”. In: *Int. Arch. Photogramm. Remote Sens. Spat. Inf. Sci* (2019), pp. 575–578.
- [22] K. Simonyan and A. Zisserman. *Very Deep Convolutional Networks for Large-Scale Image Recognition*. 2015. arXiv: 1409.1556 [cs.CV].
- [23] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. *Rethinking the Inception Architecture for Computer Vision*. 2015. arXiv: 1512.00567 [cs.CV].
- [24] M. Tan and Q. V. Le. *EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks*. 2020. arXiv: 1905.11946 [cs.LG].

4 Expanding Deep Learning for AVL

4.1 Introduction

In Chapter 3, it was shown that a pure deep learning approach is viable for aerial geolocalization. Although this is an important result, many more questions must be answered before practitioners, industry, and other organizations will want to invest in their own models. In this chapter, a series of different experiments are performed aimed at further explicating the practical properties of deep learning approaches to AVL, with the aim of improving our understanding of how these models operate, what characteristics are important to their performance, and how they can be improved. This chapter is divided into 6 different sections. These are:

- Network Architecture and Training Approaches
- Network Parameter and Region of Interest Scaling/Specificity
- Training Data Requirement
- Model Distribution Calibration
- Occlusion and Orientation Error
- Direct Raster Training Pipeline

4.1.1 Methods Applicable to All Experiments

A variety of different deep learning experiments were performed on the geolocalization dataset introduced in Chapter 3. Because of the overlap in methods, for conciseness the baseline model in use across all experiments is described here and is used unless otherwise noted. The baseline configuration uses the same data and preprocessing pipeline as in Chapter 3, and the same Xception based model with distributional outputs. Hyperparameters and settings: Input size 224x224 pixels and 1500mx1500m real width, WaCo15 test set with 1500m edge validation padding, all other datasets training set; batch size 128, Adam optimizer with $2e-4$ initial learning rate, dropping % every 250 epochs¹, $1e-5$ L2 regularization

¹this is a very minor amount of decay, following the suggestion that due to Adam's adaptive gradient estimation learning rate decay is not as necessary or effective

and $1e-3$ L1 regularization on all applicable layers except the final output layer, .5 dropout in MLP head, batch size 128, best model reported based on validation set RMSE. Outputs were independent normal distributional layers for the x and y coordinates trained with negative log-likelihood loss and a clipvalue of 1. Default training time was up to three days on a single node at the Arkansas High Performance Computing Center’s GPU72 partition or until convergence (this is more than adequate time for convergence, as most models tested here converge well within 24 hours). At time of training, one node uses a single NVidia V100 Tesla GPU and two Xeon Gold 6130 processors. More details on the model architecture and training setup can be found in Chapter 3’s methods and appendix – any details omitted here can be assumed to match the configurations used in Chapter 3.

4.1.2 Interpreting Results Between Experiments

Unfortunately, several experiments in this section were affected by a coding error which caused labels in the training set to have a bias not present in the validation set. This error causes the label adjustment in the y coordinate during the cropping step to be in the wrong direction, introducing a positive error to the y label on the training set that is uniformly distributed from 0 to 1500 meters, or an average shift of 750 meters north ($\sim .0219$ in the converted unit space). Since the validation set is not cropped, it is not affected. This discrepancy between the training and validation set necessarily implies an RMSE accuracy cap equaling the error. Fortunately, the experiments in this section are interested in comparative rather than absolute results and use the same setup within experiments. Meaningful conclusions can still be drawn between two different models equally affected by the bug. However, caution must be taken in comparing results between experiments if one is affected by the bug and the other is not. Throughout this section, experiment names that are marked with a † are affected. Additionally, depending on the aim of the experiment models were either run for a limited number of epochs or short wall clock², or were allowed to run until convergence. Using wall clock time or a limited number of epochs allows networks that train more quickly

a chance demonstrate this benefit, which is hidden when training to convergence. Contrast-

²Using wall clock time does introduce an additional external variable. Depending on demand and which nodes are in use, different HPC runs, even those performed at the same time, may have some differences in computational resources available and may therefore run at different speeds. From personal experience, however, these differences are usually small.

ingly, training to convergence allows “slow and steady” models that ultimately train to higher accuracy to show their strength. Both aspects are important to practitioners and are worth investigating – however it is important to not compare models run for a limited time for the purpose of comparison to models that were run to full convergence for the purpose of assessing maximum performance.

4.2 Network Architecture and Training Approaches

4.2.1 Introduction

Neural network architectures are extremely variable, with new architectures and paradigms constantly being introduced. In Chapter 3, a variety of historically successful CNN architectures were compared, out of which Xception [3] seemed to provide the best results on the aerial geolocalization task. Although Xception gave good results and was superior to several other models, due to time and resource limitations only a small number of different architectures could be tested, and opportunities for tuning were limited. Thus, it remains an open question whether there are other, more effective or efficient architectures for this problem. It is especially interesting to see if new paradigms in computer vision, such as transformer-based models like Vision Transformer[7], will also prove effective. Another popular newer model is the improved version of EfficientNet[19], EfficientNetv2 (or EffnetV2), which seeks to mitigate issues in the original Effnet, particularly with regard to training time[18]. Additionally, given that Xception was designed for ground-level classification problems, it is likely that it is still unoptimized for aerial geolocalization. In particular, the multiple pooling layers of Xception might make it difficult to effectively transmit location and fine feature information through the convolutional layers of the network. AVL differs from ground level classification tasks in many ways, not least of which is that the precise location of features in the input image is extremely important for AVL while it is often unimportant for RVL. This has led to an intentional cultivation of translational invariance in CNNs for ground-based classification tasks, under the assumption that translated or shifted imaged should largely elicit the same response from the neural network. In AVL, however, since translations directly alter the location of objects in the image and hence the locational labeling of the image, translational invariance is undesirable. The degree of translational

invariance in many CNN models can make them unexpectedly fail at even trivial problems like regressing the x and y pixel coordinate of a single white pixel, but this can be alleviated by explicitly encoding positional information in the neural network, as used in the CoordConv[12] approach. There is therefore a possibility that the Xception architecture can be adapted to make it more effective by either reducing positional information loss or including additional position information in the model.

Alongside new architectures, recent years have also seen the introduction of innovations in training techniques. AdamW[14] is an adjusted version of the Adam cite optimization algorithm that incorporates built-in weight decay that is decoupled from the algorithm’s adaptive per-parameter learning rates. Learning rate schedules like one cycle[17] or other cyclical learning rates have also been shown to reduce training time and increase performance on many problems. Even something as fundamental as the ReLU activation function has seen improvement, with the everywhere differentiable GELU[9] becoming the activation function of choice for many groundbreaking models, including the famous GPT-3[2] and BERT[5] language models. Therefore, AdamW and GELU are also tested as possible improvements.

4.2.2 Methods

The following subexperiments were performed according to the described methods.

Baseline: This was the baseline Xception model using all settings from above, for comparison.

No Global Average Pooling: As baseline, except that the Global Average Pooling layer before the MLP head was replaced by a flatten layer instead (which does not reduce the outputs of the convolutional layers). To maintain rough parameter parity and keep the model tractable, the number of hidden units in the MLP head had to be reduced to 240 from 4096.

CoordConv: The Xception model, except that all 2d convolution layers input had CoordConv filters added to them.

CoordconvNoPool: The CoordConv Xception model with removed average pooling as in the No Global Average Pooling model.

EffnetBase: Tensorflow’s (TF version 2.9.0) included implementation of the Effi-

cientNetB4 model. The B4 size was selected as a basis for comparison as it is the closest in parameter count to Xception.

Effnetv2Base: Tensorflow’s EfficientNetV2S model. As with EfficientNet, EfficientNetv2 comes in several different configurations with varying parameter counts. The S size model (22 million parameters in the convolutional layers) was selected as a basis for comparison it is the closest in parameter count to Xception.

Effnetv2Coordconv: The EfficientNetV2S model, as above, with input and output block 2d convolutional layers having CoordConv layers added in front. Squeeze and excite block convolutions were not modified, as although these are technically 2d convolutions they are better understood as channel pooling/weighting operations.

Vision Transformer: This test uses the Vision Transformer[7] architecture. The specific implementation can be found at [20], accessed 3/20/2022. Input configuration was selected based on the original publication and to maintain rough parameter parity with the Xception model: patch_size=28, num_layers=6, d_model=1024, num_heads=16, mlp_dim=2048, dropout=0.1. No class pooling token.

ViTCoordconv: ViT does not use 2d convolutions, however CoordConv layers can still be applied after the input to add additional location encoding to the network, which is done here. Input configuration as above.

GELU: All Relu activations in the Xception architecture and MLP head were replaced with GELU activations instead.

Xception Strided: All max pooling layers in the Xception model were removed, instead being replaced with increased stride in the preceding convolution layers, following the ideas of CITE

Xception Average Pool: All max pooling layers in the Xception model were replaced with average pooling layers.

Increased Resolution: Increased the input resolution of the images from 224 to 299 (which is the originally published Xception input size).

EffnetV2 Retuned: The EfficientNetV2S model as above, but with initial learning rate reduced to 1e-4 (from 2e-4), and weight decay parameters decreased by a factor of 10 to 1e-6 L1 and 1e-4 L2.

Experiment Name	NLL	RMSE
Xception Baseline	-6.1296	.0047
Xception NoPool	-3.5826	.0418
GELU	-5.7968	.0071
CoordConv	-6.0429	.0050
EffnetBase	-4.4474	.0280
EffnetV2	-4.9261	.0080
EffnetV2 CoordConv	-5.1013	.0075
ViT*	12.7244	.1932
ViT CoordConv*	25.5106	.1480
Xception AdamW [†]	-4.8077	.0283
Xception Avg. Pool [†]	-4.8937	.0263
EffnetV2 Tuned	-5.8576	.0044
GELU Run Tuned	-6.3626	.0055
Xception Strided [†]	-4.7526	.0271
ViT CC AdamW**	1.1599	.2297
Increased Resolution [†]	-4.7637	.0268

Table 4.1: Table of Results for the Architecture and Training Methods experiments. *Both ViT models severely overfit. ViT had training loss and RMSE of NLL: -6.2845 and RMSE: .0170. ViTCC NLL: -7.1584 - RMSE: .0082. **Although still overfitting, the performance of the ViT model degraded to a NLL -1.1887 RMSE .1438 on the train set in the second run.

GELU Retuned: Gelu, as above with the same changes to learning rate and weight decay.

ViTCCAdamW: The CoordConv ViT experiment was repeated, but trained with the AdamW algorithm and a cyclical learning rate, and a dropout of .3. Trained for 500 epochs. Initial minimum learning rate 1e-4 rising to a maximum of 1e-2 over a triangular cycle of size 120,000³ steps. 1% decay in learning rate per cycle.

4.2.3 Results and Discussion

The results of the experiments described above are summarized in Table 4.1. Results are reported in Negative Log-Likelihood (NLL) and Root Mean Squared Error (RMSE) over the validation set.

The most immediate finding was that no method definitively dethroned Xception, but there is a contender. The original Effnet continued to lag behind, and EffNetv2 with the baseline settings also underperformed. It was observed that EffnetV2 had converged very early, indicating overregularization and too high learning rate, so these were retuned downward for a second run. Retuned, EffnetV2 showed results that were comparable to the baseline model. Since the overall configuration was tuned on Xception, this is a very favor-

³This value is larger than intended - it should be divided by the batch size of 128.

able finding for EffnetV2. The original EfficientNet was designed to maximize a particular definition of efficiency based on parameter count. As the authors noted in their followup paper introducing EfficientNetV2, pursuing only parameter count was too shallow a measure of efficiency. In particular, EfficientNet utilizes specialized convolutional layers that, while parameter efficient, are slow to compute. EfficientNetv2 was designed with a more holistic view of efficiency, including training time. Here, EfficientNetv2 had an average epoch time of around 200 seconds compared to the baseline model’s $\sim 250^4$. Vision Transformer actually trained even faster, at ~ 150 seconds per epoch, despite having a slightly larger parameter count than Xception, demonstrating the transformer architecture’s reputation for ease-of-parallelization in training computation. Unfortunately, ViT’s performance did not match its speed. The presence of overfitting in the initial run inspired the followup experiment using AdamW, which implements decoupled weight decay for regularization. However, without overfitting the performance collapsed, indicating a limited ability to generalize on this problem. Considerable effort at tuning the hyperparameters of both AdamW and ViT did not improve performance notably past that reported here (results not shown). This was a surprising result given ViT’s good results on classification problems. A bug in the configuration of the AdamW experiments meant that the cycle of the cyclical learning rate was larger than intended, but it is not clear if a smaller cycle would have helped performance. After all, 1cycle[17], which achieves good results, has only one cycle per epoch. However, 1cycle has a different cycle pattern than the triangular one used here – although both use an initial increase followed by an eventual decrease.

It was notable that CoordConv improved the performance of ViT but not any other model it was tested on. For the CNNs, this indicates that positional information is being effectively propagated through the network even without CoordConv. This challenges the hypothesis that CNN performance was being limited by an inability to encode and retain detailed spatial information. While it may still be true that spatial information is lost by many operations throughout Xception’s architecture, this at least does not seem to be limiting performance. That CoordConv improved ViT’s performance is not surprising. The defining feature of transformer models is the attention mechanism which compares tokens

⁴As mentioned, variability in the HPC’s compute nodes makes this necessarily an informal comparison

to each other to generate context. However, without a form of positional encoding, the relative locations of the tokens are lost. This is precisely why the authors of the original ViT paper included a specialized positional embedding in their latest followup[1]. CoordConv fills a similar role in a simpler way by attaching positional channels to each patch/token directly. The stride and average pooling experiments were inspired by the same hypothesis that allowing additional information through the network, especially positional information, may improve performance. However, the results were consistent with that of CoordConv – no improvement – further indicating that loss of positional information is not a limiting factor of the model. The NoPooling model had noticeably degraded performance, but this is not surprising given the massive decrease in the MLP hidden unit size necessary to keep parameter parity. However, the results of the parameter scaling experiment (reported ahead) indicate that the model is massively overspecified and that hidden unit parameter loss alone cannot explain the amount of decreased performance. With this in mind, perhaps the greater reason for the decrease in performance was simply the loss of the global average pooling itself. Pooling not only keeps the network size small, but it selectively removes less important information and activations, effectively filtering out noise and redundant information. In the future, rather than reducing, it might be more effective to increase pooling for AVL deep models. GELU did not seem to improve the model, though it did change the dynamics enough to require some hyperparameter tuning. The first run using GELU converged early, so as with Effnetv2 it was rerun with new parameters. Once the early convergence was addressed, its performance improved to nearly match the base Xception model. This is not sufficient to recommend GELU for AVL, since GELU is computationally more expensive than ReLU activation. Surprisingly, increasing the resolution of the input image did not improve performance at all – in fact, performance seemed to slightly decrease. Although it might seem intuitive that increasing the amount of information in the input should increase performance, there are many reasons why this might not occur. One possible reason was already discussed in Chapter 3: it is likely that the most permanent, and therefore most identifying, features of an aerial image are large. In addition, many small features like cars and people frequently change and therefore their inclusion in the input makes generalization more difficult. As increasing the input resolution also increased the amount of high-frequency

features in the input, to the extent that these features are transient their inclusion can only help the model overfit, not generalize to unseen data. This finding suggests that increasing input resolution is not an effective approach to improving deep models on AVL, and may actually be detrimental. A next step would be to see how much the input resolution could be decreased without hampering performance.

4.3 Network Parameter and Region of Interest Scaling/Specificity

4.3.1 Introduction

Deep networks can be very efficient at storing complex information and relationships in their parameters. However, neural networks are not infinite in their storage capacity, and as they are required to encode larger amounts of information they must increase in size. Hoffman et al. [10] that, at least in the context of large language models, the amount of training tokens and the parameter size of the model should scale equally for compute-optimal training. In Chapter 3, it was hypothesized that the architecture used was larger than necessary for the geographic size of the region of interest. To investigate this claim further, the model was tested with reduced parameter count and with a reduced region of interest (reducing the region of interest increases the ratio of parameters to geographic area without intractably increasing the network size). That is, the aim is to investigate the effect of the ratio of model size (as measured by parameter count) and the geographic area aiming to be localized over.

Another question relating the real-world characteristics of the input data and the effective of the deep geolocation model is whether different types of terrain are more or less difficult to localize. It is likely that regions without distinctive, permanent features (such as open ocean) will be much more difficult to localize over. Although it is beyond the scope of this investigation to assess the difficulty of localization over all possible biome and terrain types, Washington County contains a varied mix of rural, suburban, and urban developed areas. Thus, a brief experiment was performed to see if different regions of the same geographic area would show significant variance in validation accuracy.

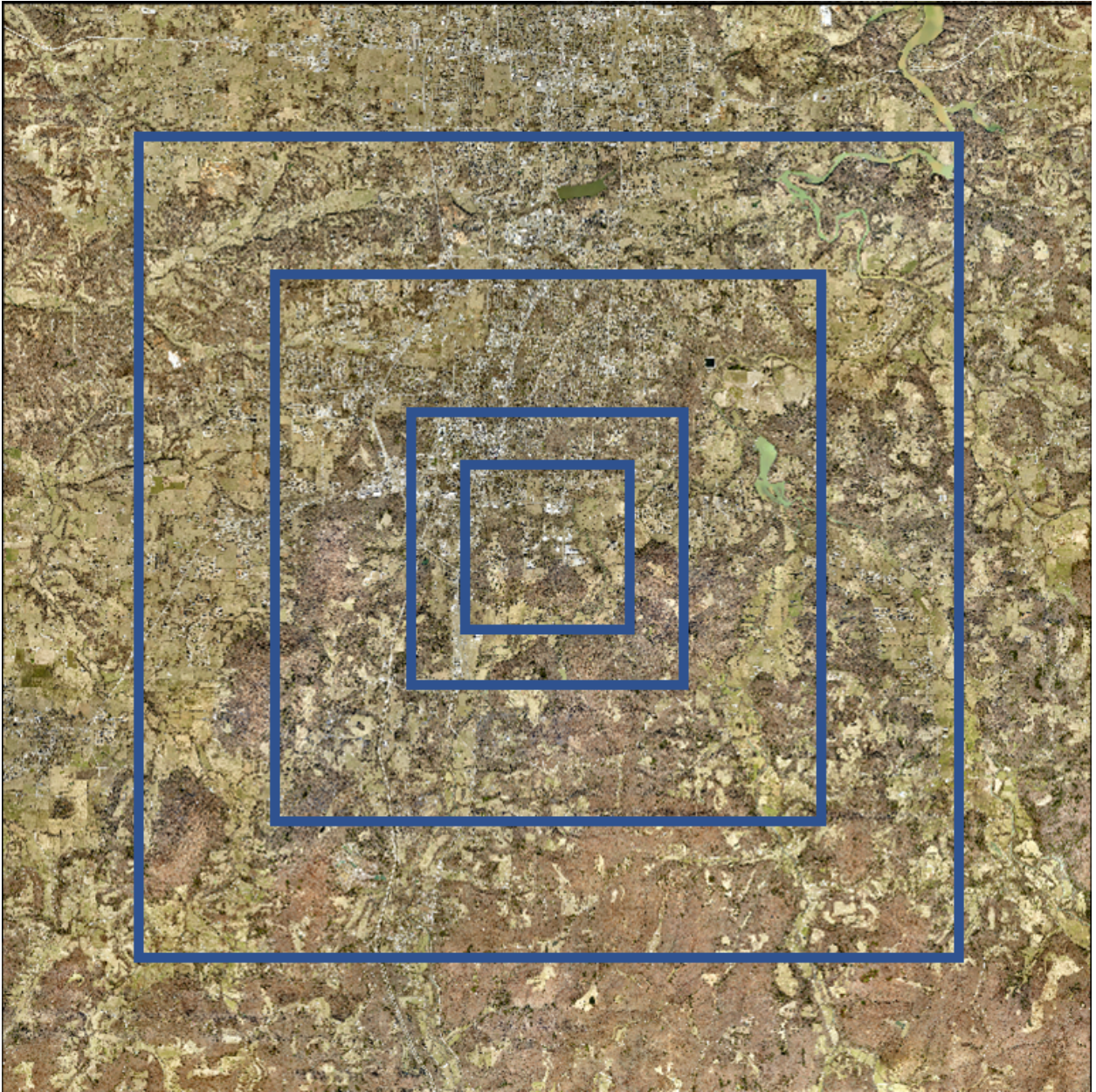


Figure 4.1: Figure illustrating the RoIs for the reduced RoI experiment. The background is the original RoI, while each blue square represents one of the scaled down RoIs. From smallest to largest, the scaled down RoIs have widths of .15, .25, .5, and .75 of the original RoI.

4.3.2 Methods and Results

The model was applied to train and test sets on a restricted region of the original RoI. Valid padding was kept constant at 500m to avoid edge effects, which would become more prominent for smaller regions as a greater percentage of the total area is near an edge. Learning rate decay was not used, as the different sized regions have very different epoch sizes. The setup for the data is shown in Figure 4.1. The results are shown in Table 4.2.



Figure 4.2: Figure illustrating the RoIs for the specific region experiment.

Experiment Name	NLL	RMSE	Scaled RMSE
.15	-7.1826	.0012	.0080
.25	-7.5376	.0019	.0076
.50	-6.5951	.0043	.0086
.75	-5.9168	.0073	.0097
1.0	-5.3008	.0109	.0109

Table 4.2: Table of results for RoI scale comparison.

For the region specificity experiment, the RoI was divided into a 3x3 grid. A baseline Xception model was trained over each grid square. Grid locations are as shown in Figure. Edge padding for the validation set was 1000m. No learning rate decay was used. Figure 4.2 shows the partitioning of the RoI. Results are shown in 4.3.

Grid Location	RMSE
1	.00073
2	.00083
3	.00082
4	.00079
5	.00077
6	.00081
7	.00079
8	.00080
9	.00082

Table 4.3: Table of results for the different grid locations.

In order to investigate the relationship between model size and performance, several downscaled models were trained over the full RoI. Models were downscaled by multiplicative factors reducing the number of filters in convolutional layers, and hidden units in dense layers (except for the output layer). Note that all models were affected by the y-coordinate error bug described in the general methods. Results are shown in table 4.4.

Filter Scale	MLP Scale	Params	NLL	RMSE
1	1	45,997,356	-4.8280	.0262 [†]
.75	.75	25,975,034	-4.7525	.0267 [†]
.5	.5	11,607,576	-4.6343	.0280 [†]
1	.5	29,207,852	-4.6917	.0284 [†]
.5	1	26,272,600	-4.7083	.0271 [†]
.0625	1.0	17,424,055	-3.2428	.0530 [†]
1	.0625	21,452,844	-4.0264	.0298 [†]
.0125	.0125	754,571	-4.0849	.0309 [†]
.0625	.0625	201,315	-3.1389	.0521 [†]

Table 4.4: Table of results for parameter count scaling experiment.

4.3.3 Discussion

RoI Downscaling

At first glance, the finding that accuracy improves as the ROI decreases might seem to indicate that the model is being limited by its ability to encode the entire region, and reducing the size of the region is freeing up this bottleneck. However, in smaller regions the overall scale of the labels is reduced. At half the width, the average distance between any two labels is also halved. Thus as the scale of the label space decreases we should expect to see a proportional reduction in error size. Because of this fact, the normalized distances per region are also reported, which allows direct comparison of the relative error. When the scale of the errors is standardized in this way, it is apparent that accuracy is not being limited by the size

of the RoI being too large in the baseline model. The very slight decrease in performance on the two largest regions seems to be due to the 12 hour time limit on training – training takes longer to converge on the longer regions, and these models had not quite finished converging. An interesting follow-up experiment might be to measure how quickly a model converges over a region of a given size, or with a given number of parameters. In this experiment, the smaller regions and models converged noticeably more quickly, which is expected. However, to truly quantify the difference in convergence would require a specialized setup that equalized other time affecting factors in training. For example, in the current setup because the number of image chips within smaller regions is reduced, the size per epoch is also reduced. This increases the time cost of between epoch operations such as shuffling and saving. The overall effect of the epoch-based learning rate decay schedule is also dramatically increased when epochs are small and short. A quantitative estimate of how model convergence time scales with region size would certainly be of interest to anyone looking to build a production level AVL model.

Region Specificity

It was expected that the specific region experiment would show some variability in results. Because the middle and top middle grid square, encompassing Fayetteville and Bentonville, are far more developed than the rest of the RoI, it was thought they would show different results from the more rural squares. However, the results were quite similar across all 9 subregions despite large differences in level of development. This seems to indicate that, at least for the sort of undeveloped terrain in the mountainous Washington County, less developed areas are not more difficult to localize.

Parameter Reduction

The model proved to be remarkably robust to loss in parameter count, indicating that it was heavily overparameterized in its default configuration. This is a favorable finding for deep learning applications to aerial geolocation, as it indicates that the size of the model can be considerably smaller with minimal loss in performance, allowing for faster training and inference as well as deployment on more platforms. Alternatively, a far larger region of

interest can be encoded by our baseline model. Performance only noticeably degraded at very high reductions in filter or hidden unit count. Overall, the model was slightly more sensitive to loss in filter count than to loss in hidden units, indicating that the convolutional portion of the network is more important than the MLP head. This is consistent with the literature – it is even possible to completely dispense with the MLP head, creating what is known as a fully convolutional network[13], while still obtaining good results. Future work could see if this approach would be effective for AVL.

It does seem to be the case that NLL was more sensitive than RMSE. A model with the same RMSE but higher NLL is less well calibrated. The likely cause of this is that since the NLL is much more sensitive to the estimate of mean than the estimate of standard deviation, when parameter resources become constrained, performance on estimating scale is sacrificed to increase performance on estimating the mean. This is not a bad property to have, since the accuracy is indeed more important than calibration (you can get perfect calibration with no dependence on the data – estimate the mean every time and set the standard deviation to the population standard deviation – but this is obviously not a useful model). See the Model Distribution Calibration experiments in the next section for further discussion.

4.4 Training Data Requirement

In the experiments in Chapter 3, all available data was used, and a good result was obtained despite the relatively low number (13) of total datasets. Obtaining and curating data is often the most challenging and most important part of training a deep neural network. Thus, data efficiency is of paramount practical importance. To this end, this experiment investigates how geolocalization accuracy is impacted by restricting the amount of training datasets. It further investigates how similarity of the training and validation data impacts accuracy. For example, whether or not additional datasets in the training set of a different foliage season (that is, leaf on or leaf off) from the test set impact accuracy on the test set.

4.4.1 Methods and Results

A total of eleven models varying the training and test sets were trained. All settings were in the default configuration described, except that L1/L2 regularization was not added to the Xception layers⁵. Models were run for 3 days wall time, with the lowest RMSE validation epoch being chosen as the final result. The experiments and their results are summarized in Table 4.5.

Exp. #	Train Sets	Test Sets	Train Fo- liage	Test Fo- liage	Train RMSE	Test RMSE
1	ADOP2017	WaCo15	Off	Off	.0484	.2702
2	NAIP2006	ADOP2006	On	On	.8620	.2519
3	NAIP2015	WaCo15	On	Off	.0211	.2642
4	NAIP2006 NAIP2015	AO15	On	Off	.0291	.2821
5	NAIP2006 NAIP2015	ADOP2006	On	On	.1443	.2539
6	NAIP2006 NAIP2015 ADOP2017 AO19	WaCo2015	2 On, 2 Off	Off	.0193	.0528
7	All WaCo sets, ADOP2017	NAIP 2015	Off	On	.0198	.2295
8	All leaf off sets, ADOP2006	NAIP2015	6 Off 1 On	On	.0195	.1155
9	All leaf off sets, ADOP2006, NAIP2006	NAIP2015	6 Off 2 On	On	.0198	.0364
10	WaCo2016	WaCo2015	Off	Off	.0189	.1556
11	NAIP2006 NAIP2015	NAIP2009	On	On	4.428	.2552

Table 4.5: Table of results for the training set experiments.

4.4.2 Discussion

In order for a model to learn to generalize effectively, it has to see at least a sufficient number of different examples with the same labels to learn some sort of generalized function connecting input to the labels. It is therefore unsurprising that in Experiments 1, 2, and 3, where there is only one training dataset, and therefore only one example for each piece of terrain, we receive a result of .25. This score is approximately the error a model would achieve by outputting the middle of the RoI - .5, .5 - on every image. This is no better

⁵There is no particular reason for this, but I audited every file in this project while writing it up and it happens to be true. The lack of additional regularization makes generalization harder and thus the results here are worse than they would be otherwise, but since the purpose of the experiment set is comparative and all the experiments have the same configuration it doesn't change any conclusions.

than a baseline mean model –there is no generalization at all. The only experiment with one training set that did better than the baseline was Experiment 10, which specifically trained and tested on the most similar pair of datasets available, but results were still poor. It is interesting that the NAIP2006 dataset also had low train accuracy as well, indicating that the dataset is more difficult to learn locations from. Two train datasets were also insufficient to learn to generalize, even if foliage matched, as evidenced by Experiments 4 and 5 and 11⁶. Experiment 6, with four total train datasets, finally shows some generalization capability – however, the model still overfits to the train datasets. Experiment 7 uses 6 datasets to train, but none are matched to the leaf type of the test set. With the color balance completely different between train and test, the model does not learn to generalize to the different foliage. The addition of even one matching foliage dataset in Experiment 8 allows the model to generalize better than baseline, but still at a poor level. It is interesting to note that this is still much better performance than Experiment 5, which had two foliage matched training sets – indicating that the non-foliage matched sets, despite having a completely different color balance, are helping the model learn to generalize given the presence of at least one foliage matched sample. This is an important finding as it suggests that we need not worry about balancing the distribution of our input datasets to our expected use – all the data should be used. The addition of one more leaf-matched set provides a massive boost in performance in Experiment 9 indicating that large gains can be achieved from the addition of even one dataset in the right circumstances. Overall, these experiments highlight and confirm the need for multiple and varied datasets to achieve effective generalization.

4.5 Model Distribution Calibration

4.5.1 Introduction

As discussed in Chapter 3, a major issue with most deep learning regression is that they only output a prediction alone, with no easy way to assess the prediction’s certainty. This is in contrast to classification where the relative scores of the predictions across classes can give a built-in measure of model certainty. A variety of methods for different use cases

⁶Experiment 11 was run as a follow-up to experiment 5, due to concerns that ADOP2006, being late summer and having browning leaves and grass, was a bad example of a leaf-on dataset. However, it turned out to make no difference even if the vibrantly green NAIP2009 set was used instead.

have been developed to try and address this problem. For example, the popular method of Monte Carlo Dropout[8] uses a model’s existing dropout layer to get an empirical estimate of model uncertainty. It has the advantage of being applicable post-hoc, but a major disadvantage is that it requires running the model inference step numerous times to generate a statistically relevant amount of samples. The approach used for the model presented in Chapter 3 was to have the output layers of the output parameters defining a probability distribution (such as the Gaussian). This approach goes by several names in the literature, being variously called a probabilistic neural network, a heteroscedastic neural network, a heteroscedastic probabilistic neural network, or simply a neural network with distributional outputs⁷. Since the model in this approach outputs parameters defining a probability distribution, rather than a single point estimate, it becomes possible to directly quantify the model’s uncertainty⁸. However, a measure of uncertainty is less useful – even misleading – if it is miscalibrated. A probability distribution is considered well calibrated if the predicted distributions closely match the true distributions of the outputs. For example, an outputted normal distribution can be considered well calibrated if, for all proportions X , approximately X proportion of the true labels are contained within an X proportion confidence interval. At a confidence level of 5%, 5% of the true labels should be inside the 5% confidence intervals of the distributions produced by the model. Likewise, at a confidence level of 95%, 95% of the true labels should be inside the 95% confidence intervals of the distributions produced by the model. While confidence intervals are an easily understandable and visualizable measure of calibration, they are difficult to calculate, or even ill-defined, for models that are nonsymmetrical or multimodal[11]. Thus, when comparing across models that have these properties, we resort to different metrics. A widely applicable one that also allows direct comparison between different distributions is simply the likelihood. This also happens to be the loss function used to train out model (in the form of negative log-likelihood), so it is used here to compare across models. The downside is that this measure does not indicate whether a model is over/underconfident.

⁷Many of these names are also used for related but distinct approaches in, for example, Bayesian neural networks. For this reason, the less ambiguous terminology of distributional outputs is used here.

⁸Naturally, this can only quantify the model’s uncertainty within the bounds of the chosen output distribution, and does not estimate higher-order uncertainties – uncertainty about the uncertainty

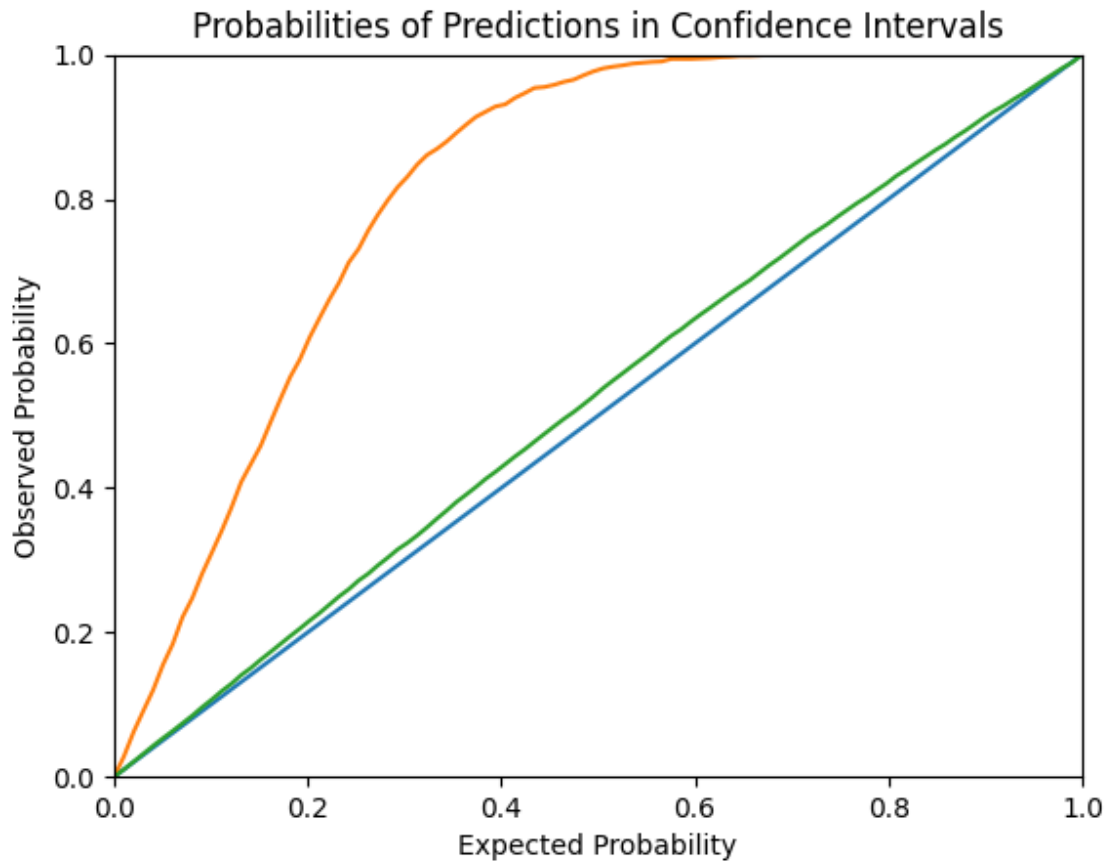


Figure 4.3: Figure illustrating the difference between the training and validation calibration. The green line is the calibration over the training set, while the orange line is the calibration over the validation set. These calibrations were taken from the baseline model in the architecture experiment.

In Chapter 3, it was found that the models produced were underconfident. Subsequent analysis showed that this underconfidence was only present on the validation data, while the model was well calibrated on the training set (see Figure 4.3). This is unlikely to be simply overfitting as the models did not show overfitting on RMSE. Furthermore, if the model was overfitting to the training data then the trained standard deviations would become small, which would produce overconfidence on the validation set instead of the observed underconfidence. An alternative explanation might be that the model’s calibration is being impacted by an overly large amount of weight decay. The model’s weight decay parameters were tuned using RMSE as the metric, without regard to the NLL. In general, the model’s mean parameters have a much greater impact on the NLL loss than its scale parameters. Thus, gradients are higher from the mean, so a level of weight decay that might not negatively impact the mean estimation – and therefore not show when model performance is measured by RMSE – might still be too high for proper calibration on the standard deviation. Several models are trained with less weight decay to investigate this possibility. Another possibility,

also investigated here, is that the larger gradients associated with the mean simply mean that the mean parameters converge more quickly than the scale parameters. Thus, models that seem to have converged in terms of RMSE might actually not have reached convergence in NLL. For this reason, longer training of models where convergence is assessed by validation NLL are tested.

Another possible source of poor calibration could be the use of a normal distribution to model the output uncertainty. Since the true uncertainty landscape of the output conditioned on the input is likely not normal, this might lead to poor calibration. For example, if there are two visually similar regions separated by geographic distance, and the true probability distribution the normal distribution cannot represent this type of uncertainty. Additionally, modeling the X and Y coordinates Gaussian distribution makes other assumptions: that the true distribution has infinite range, it is symmetric, it is independent for the X and Y coordinates, and the tails of the distribution are thin. The first assumption is certainly false, and the remaining three are likely also false to at least some extent. Of these assumptions, the only one investigated in Chapter 3 was the independence of the X and Y coordinate. Since the only the X and Y coordinate as a pair uniquely define a particular location, the true distribution is certainly not independent in the location coordinates. However, in Chapter 3 it was found that using a bivariate normal to model the output enhanced neither the accuracy nor the validation calibration, indicating that this dependence is not critical to include in the model. Whether or not the other assumptions made by using a normal distribution are negatively impacting model performance and calibration remains to be tested. Various alternative distributions are tested as possible outputs of the model, to see if they improve either calibration or performance.

Finally, if model calibration between the training set and validation set cannot be equalized during initial training, it may still be possible to mitigate the model post-hoc. A large literature of such methods has been developed, though techniques for calibrating regression models have been less investigated than techniques for calibrating classification models. Cui et al.[4] investigated several post-hoc calibration methods for heteroscedastic neural networks, ultimately recommending a second training phase using maximum mean discrepancy on samples to calibrate the model after initial training using negative log-likelihood, which

is implemented and tested in this section. Palmer et al.[16] also propose post-hoc calibration for Gaussian outputs. However, rather than modifying the entire network, they aim to only adjust the scale parameter, optimizing under the assumption of a linear transformation. Their technique is tested here as well. The assumption of a linear transformation on the scale parameter is limiting. To investigate whether alternative functional forms can produce better results, an experiment using a secondary calibration neural network is also performed, as neural networks are known to be able to express many different functional types.

4.5.2 Methods and Results

Throughout this section, the best epoch was selected based on validation NLL loss as opposed to the validation RMSE as in other experiments, as the aim was to analyze overall calibration over accuracy. However, since NLL is a combined measure of accuracy and calibration, it is usually the case that the best RMSE and the best NLL epoch are the same or very close.

Methods: Several models were trained with different distributions being predicted by the output layer. These include the normal distribution (same as baseline), a truncated normal, a truncated Cauchy distribution, a Laplace distribution, a mixture distribution of two normals, and a mixture distribution of three normals. In addition, the Xception model was trained with low regularization (reduced L1/L2 regularization by a factor of 10, .3 dropout) and very low regularization (no L1/L2 regularization, .1 dropout).

A model was also trained using the AdamW algorithm, which is simply the Adam optimizer with weight decay. The AdamW model uses a cyclical learning rate schedule with a triangular shape, starting at the minimum learning rate. A factor is chosen to multiply the initial learning rate by to determine the max learning rate of the cycle. The tuned values and ranges were as follow: learning rate, 1e-6 to 1e-3 with log sampling; learning rate factor 1 to 100, log sampling; weight decay 1e-7 to .1, log sampling, cycle length 60000 to 260000, with step size 40,000⁹ The values returned by the search and used in the model were: initial learning rate 3.866e-4, max learning rate 4.298e-3, cycle length 14,000, weight decay 2.169e-

⁹These values should have been divided by the batch size but were not. Thus, they are all 128 times larger than intended and the cyclical learning rate only completes a few cycles over training.

6. Because the AdamW model has its own weight decay, L1/L2 regularization is removed from the Xception model. L1/L2 was also removed from only the Xception filters during continued training, under the hypothesis that reduced regularization might be helpful for calibration.

The Baseline model from the Architecture Experiment is loaded for continued training using either the same Adam optimizer as baseline or the Maximum Mean Discrepancy loss according to the procedure described by Cui et al.[4]. Learning rate was increased to 1e-3 and weight decay parameters were decreased by a factor of 10 to compensate for the lower scale of MMD loss.

The results of all these configurations are summarized in Table 4.6

Exp Name	NLL	RMSE
Normal [†]	-4.8376	.0252
TruncNorm [†]	-2.2282	.0275
TruncCauchy [†]	-1.96160	.0263
Laplace [†]	-2.2301	.0246
Mixture2 [†]	-5.0827	.0269
Mixture3 [†]	-5.4924	.0266
LowReg [†]	-5.4217	.0268
Very Low Reg [†]	-5.7685	.0276
AdamWTuned [†]	-4.8884	.0251
Cont. Training*	-2.4136	.0303
MMD Post-Train*	9.67e9	.2458

Table 4.6: Table of results for calibration experiments. *The continued training experiment produced a severe degradation of the saved model in the first epoch.

The results of the lower regularization runs are summarized in Figure ??

Two post-hoc calibration techniques were tested. One is based on Palmer et al. [16] using standard optimization techniques to fit a linear transform that minimizes calibration error, the other is an original approach that fits a secondary deep model to attempt to transform the standard deviations to minimize negative log-likelihood. First, the predicted means and standard deviation from the baseline model in the architecture experiment on the validation set are split 70-30 into a train and test set. Then, in the first method, the parameters of a linear transformation on the standard deviations are optimized using the Nelder–Mead [15] method on the calibration error¹⁰. In the second method, a small neural network is trained minimizing the negative-log-likelihood over a normal distributional output,

¹⁰In this case, simply the area between the observed calibration and the ideal calibration on the calibration curve.

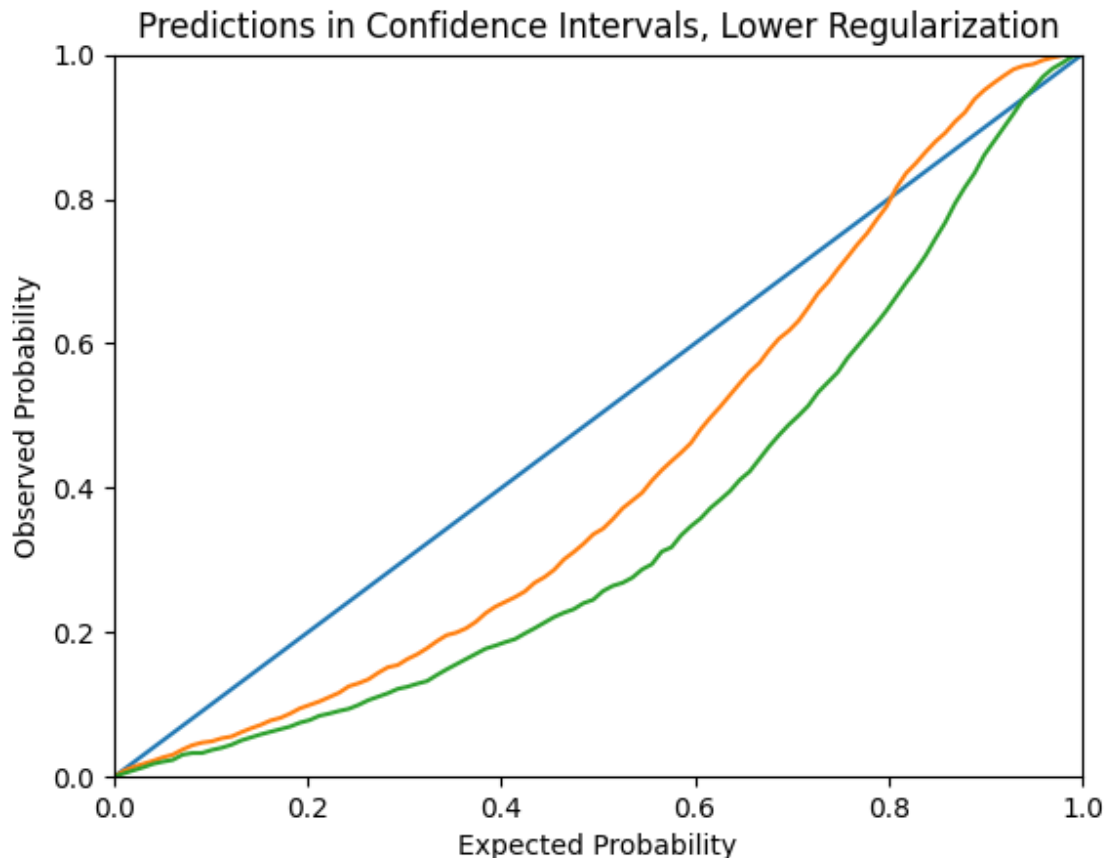


Figure 4.4: Figure illustrating the effect of lowering regularization on the calibration. Calibration for the validation set is shown for the low regularization (orange) and very low regularization (green). The blue line represents perfect calibration.

with the means of the output forced to be the means of the input (so that only the standard deviation can be trained). This network takes as inputs the means and standard deviations of the AVL model, and outputs new standard deviations. The results are summarized in Figure 4.5.

4.5.3 Discussion

The two techniques that improved calibration as measured by NLL were lowering the regularization and the use of the multi-modal mixture density distribution. This provides support for the hypotheses introduced in the introduction of this section that regularization impacts scale parameters more, and that the ability to model multi-modal uncertainty would help overall calibration. It's worth noting that the lower regularization models went from being mostly underconfident to mostly overconfident, with the very low regularization model having worse overall calibration. This suggests that regularization techniques should be tuned carefully to avoid miscalibration. The other models generally had worse NLL than the baseline model. This is not all that surprising for the Cauchy and Laplace models, as

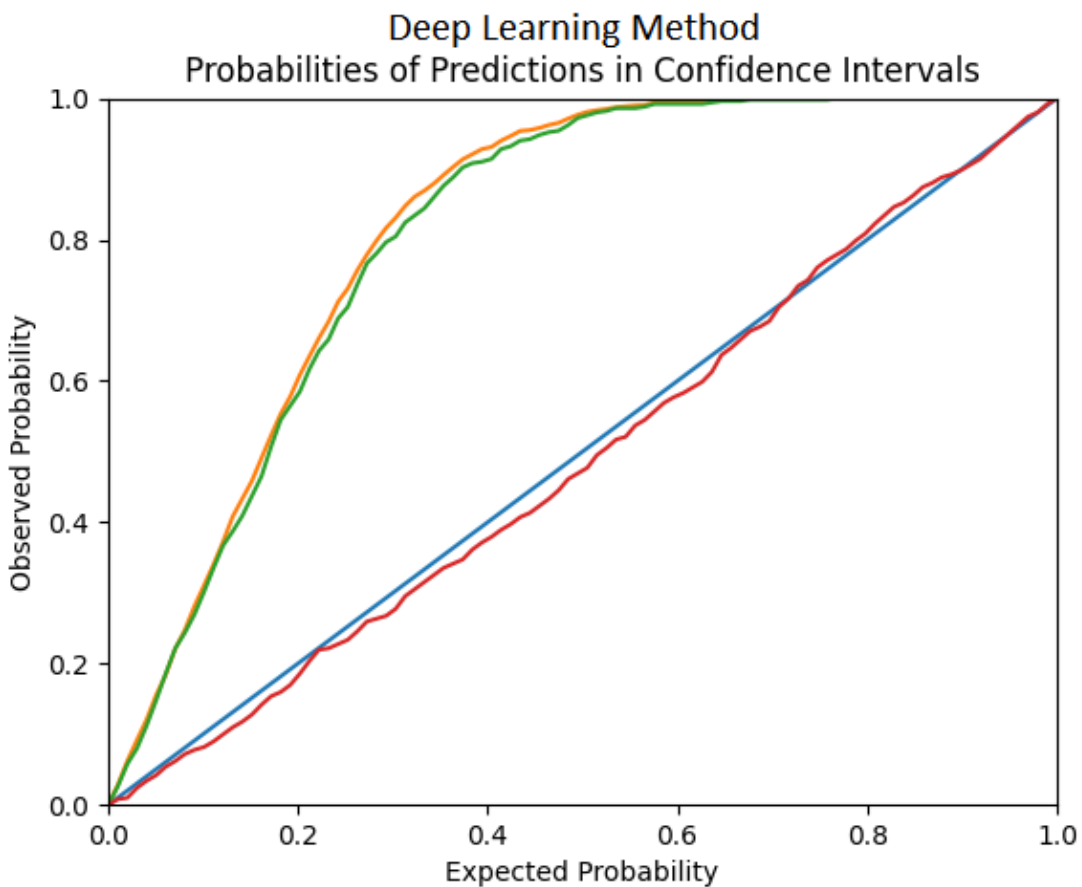
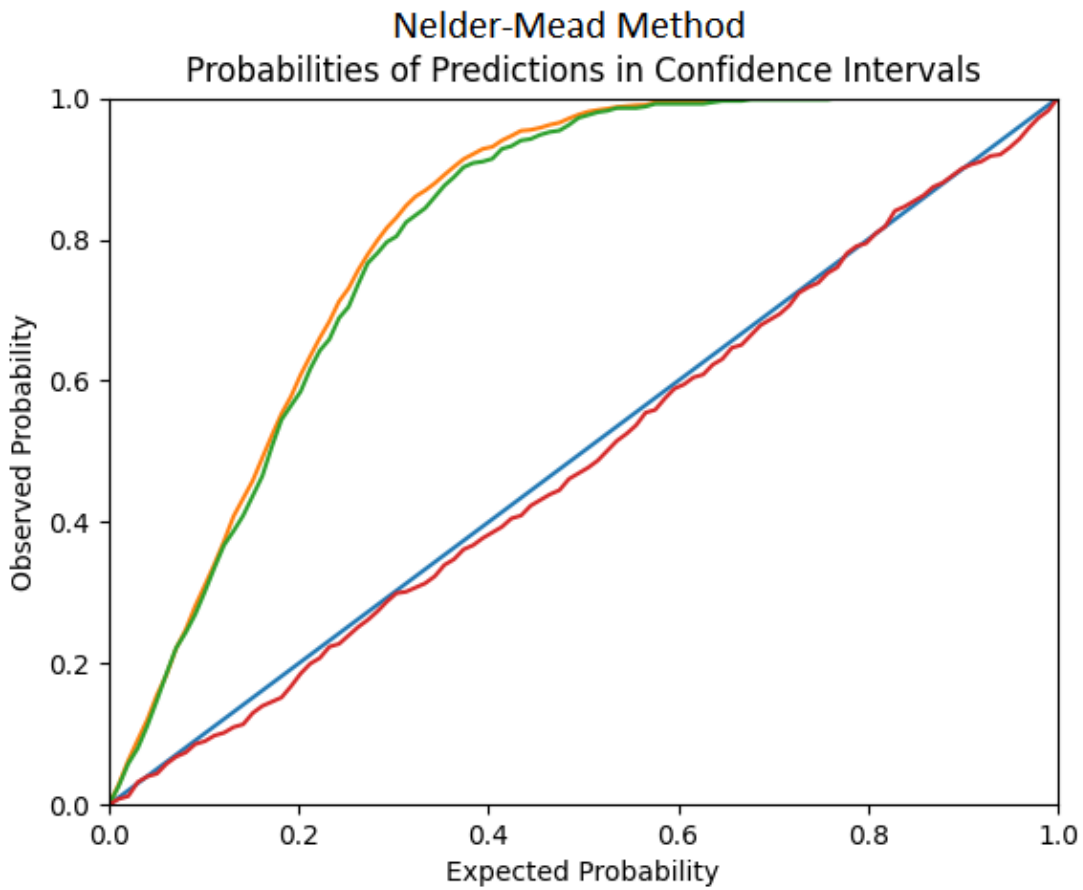


Figure 4.5: Figure illustrating the relative effectiveness of the two post-hoc calibration methods. The orange line is the calibration across the entire validation set. The green line is the calibration across the part of the calibration set withheld out for testing. The red line is the calibration of the withheld testing set post-recalibration. The blue line represents perfect calibration.

although they have fat tails (which is why they were selected for testing), they are distributions that don't arise naturally as often as the Gaussian. The fact that the truncated normal distribution did so poorly, however, was unexpected and warrants further investigation.

Continued training using the same settings (sans regularization on the filters) did not improve calibration, indicating that the previous models had indeed reached convergence even on the scale parameters. With the above results on regularization, it is likely that the baseline settings did not allow for good calibration at convergence, and this remained true even with the L1/L2 regularization only being applied on the MLP head.

Cui et al.'s idea of post-training on MMD loss to directly optimize for calibration is intriguing, however it has a major flaw. While training on MMD loss, there is no direct factor encouraging accuracy, and the post-training can and will alter the model's mean predictions as well as standard deviation. On an overconfident model, this might not be too catastrophic – the only two ways to improve the MMD loss are to either improve the prediction or to better calibrate the scale, both of which are favorable. Our models, however, are underconfident. And there is a very easy way to make underconfident models better calibrated – decrease their accuracy. From the perspective of pure MMD loss, there is no difference between adjusting the scale to be well calibrated with the predicted means, and adjusting the means to be well calibrated with the predicted scale. The fact that the RMSE loss is the baseline value of .25 suggests that the MMD training has accomplished exactly that – by outputting the mean of the input distribution always regardless of input, all the model has to do is estimate the population standard deviation every time and it will be perfectly calibrated. Clearly, however, this is not a worthy trade off.

The issue of post-training destroying the estimates to achieve better calibration can be easily solved, however, if we lock the location estimates and only adjust the scale parameters. This is exactly the approach taken by Palmer et al [16]. They assume a simple, fixed linear transformation of the scale parameter can improve calibration, and then just use the Nelder-Mead [15] algorithm to optimize the parameters of the linear transformation. The top of Figure 4.5 shows the results of this methodology when applied to our own ill-calibrated model, which is very good. However, it is possible that there is some nonlinear transformation that might perform even better. To investigate this, an experiment was

performed that used deep learning to find an appropriate transformation for the standard deviation to improve calibration. In this approach, a secondary deep model is trained to adjust the scale parameter. Using the predictions of the baseline model as inputs, it is tasked to transform only the scale parameter to produce an output normal distribution that minimizes the NLL. This model also has the advantage of seeing the estimates for both x and y in the inputs, so it can potentially model and exploit covariance. The bottom of Figure 4.5 summarizes the results, which were very good and quite comparable to the linear transformation approach. However, the linear approach is simpler and easier to compute, and so is recommended. One remaining question is whether this sort of optimization would work equally well on validation sets containing multiple datasets, or if they would require separate tuning per dataset. Overall, these experiments show that the calibration of the model can be substantially rectified at low cost, with positive overall implications for deep approaches to aerial geolocalization.

4.6 Occlusion and Orientation Error

4.6.1 Introduction

The data for the AVL came from survey flights which were all performed in clear weather conditions. Clearly, in practice we cannot always rely on good weather conditions. Clouds commonly obscure portions of the ground from an aerial vantage. Thus, the robustness of the deep learning based geolocalization model to visual obstructions such as clouds is of great practical importance. In addition, an aerial platform will take pictures at many different orientations. While these can then be processed into a canonical (e.g. north-south) orientation, the correction may be imperfect (due to, for example, inaccuracy in the compass which can have numerous causes). Thus, investigation of the robustness of the model to rotational perturbation is also of interest. The removal of rectangular sections from input images in deep learning is also known as Cutout [6]. Cutout is a method of regularization for image datasets, designed to be an image analog of dropout, where rectangular regions are removed from the input images at random. In their paper, names showed that cutout was an effective regularizing technique, improving performance on a variety of computer vision tasks by reducing the model's ability to overfocus on small parts of an image. An important

caveat to Cutout training is that in order to see benefits the model must still occasionally see some unoccluded images. Therefore, the minimum number of rectangle cutouts was set to zero at all occlusion levels to ensure that this could occur. Note that in order for the probability of any given pixel to be covered by a rectangle to be uniform, the center of a rectangle can be outside of the image’s boundaries.

Rotation can be clockwise or counterclockwise, using bilinear interpolation, with lost corner pixels being filled black. When rotation and masking was combined, rotation was applied second. Average occlusion was determined by a Monte Carlo sample over 1,000 images. During training, validation was done on the unmodified validation set. At the end of training, the final model (based on best validation RMSE) was test on the validation set modified by the light, moderate, and severe occlusion settings.

4.6.2 Methods

The base model was modified to allow for rotation as well as the positioning of random black rectangular masks. Settings for the experiments are summarized in Table 4.7, where the side lengths are in pixels. See Figure 4.6 for a visual example of the occluded images.

Exp. Name	Max Rect.	Min Side	Max Side	Max Rot.	Average Cover
Baseline [†]	0	NA	NA	0°	0%
Light Occlusion [†]	10	20	80	0 deg	22.88%
Moderate Occlusion [†]	13	40	100	5°	42.18%
Severe Occlusion [†]	16	80	120	10°6	65.97%
Rotation Only [†]	0	NA	NA	10°	3.9%
High Rotation [†]	0	NA	NA	90°	11.6%
Full Rotation [†]	0	NA	NA	180°	11.7%

Table 4.7: Table of configurations for the occlusion experiments.

4.6.3 Results

The results of the experiments are summarized in Table 4.8. The columns represent the setting of the validation data - e.g. the row labeled Baseline and the column labeled Severe Occlusion mean the model trained on baseline images assessed on severely occluded validation images.

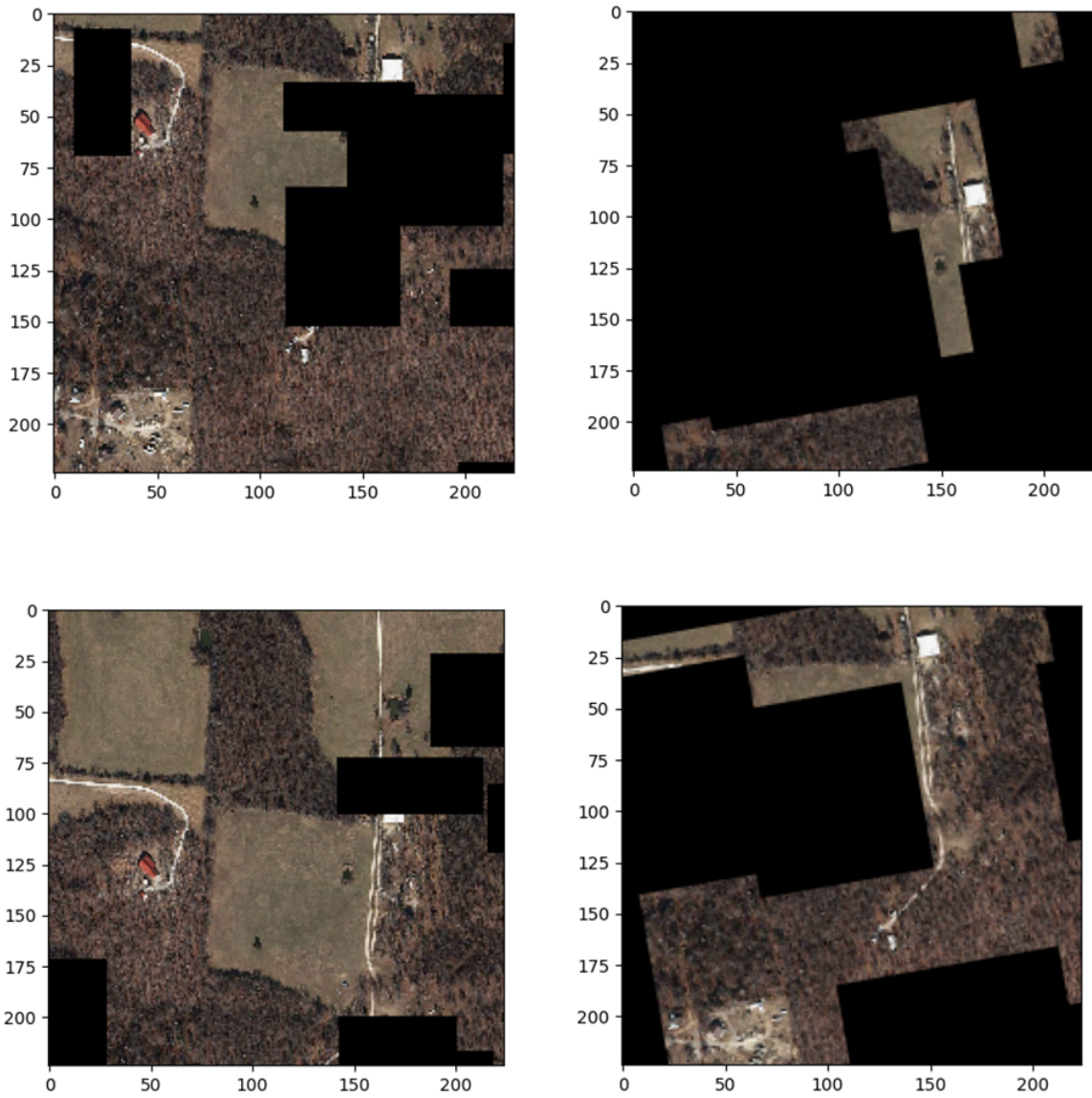


Figure 4.6: Figure demonstrating the occlusion modification. Images on the right were generated using the heavy occlusion settings, while images on the left were generated using the light occlusion settings.

Experiment Name	Baseline	Light Occlusion	Moderate Occlusion	Severe Occlusion
Baseline [†]	.02442	.2083	.2326	.2694
Light Occlusion [†]	.0252	.0497	.0911	.1722
Moderate Occlusion [†]	.02543	.0311	.0491	.1055
High Occlusion [†]	.0253	.0288	.0385	.0858
Rotation Only [†]	.0251	.1089	.2169	.3034
High Rotation [†]	.0421	.0744	.1152	.2239
Full Rotation [†]	.0766	.1228	.1458	.2011

Table 4.8: Table of results for the occlusion experiments.

Discussion The first finding is that the baseline model, which has never encountered missing data during training, is unsurprisingly brittle to even mild occlusion. However, models trained on occluded training sets were much more resilient while also not sacrificing performance on the unoccluded validation set. This is an encouraging finding as it indicates that the model can become robust without trading off accuracy. This is also in line with expectations, as the simulated occlusion is similar to applying Cutout regularization. It is especially interesting that performance by the models trained with high and moderate occlusion outperformed the model trained on light occlusion on the light occlusion validation set. Likewise, the high occlusion trained model outperformed the moderate occlusion trained model on the moderate occlusion validation set. This suggests that to improve performance on a target level of occlusion, we should actually not train on that level of occlusion, but a greater one. Overall, the fact that performance either stayed the same or improved on less occluded images when training was done on occluded images suggests that training with random occlusion should be a standard practice for AVL. The model proved to be quite robust to low levels of rotation. However, performance on the baseline validation set degraded substantially for higher rotation levels in training. This is in sharp contrast to the results for occlusion, where baseline performance was not negatively impacted by training on the augmented dataset. This is unsurprising as navigation relies heavily on the true orientation of features – if you are trying to find North College Avenue, which runs North-South, and you fly over an East-West road you know you are not in the right location. Unfortunately, this means that it is not viable to simply train on randomly rotated images and achieve robustness to poor compass calibration. On the other hand, small orientation deviations as might occur even on a well-calibrated compass will not affect model accuracy.

4.7 Direct Raster Training Pipeline

In most deep learning computer vision tasks, the typical data pipeline involves numerous individual image files each associated with a label. Following this paradigm, in the work in Chapter 3 an offline image chipping process was performed to produce small raw images which could be loaded with their associated label. It is easy to see, however, that this is inefficient and introduces additional sources of bias. It is inefficient because the preprocessing step duplicates much image information due to overlapping images, and because subsequent cropping or resizing is computationally expensive. It introduces a potential source of bias because the locations chosen for image chips during the offline step are fixed and constant in every epoch instead of being randomized throughout training, and thus certain regions will, by chance, be oversampled or undersampled consistently during training (that is, regions which by chance are included in more chips are trained on more often). Additionally, the offline chipping process makes altering the data input characteristics cumbersome. If, for example, a practitioner wanted to change some aspect of the chips, such as their orientation, they would need to re-chip every dataset, produce labels for these new chips, then transfer all of these new chips to whatever platform they are training their model on. A superior approach would be to draw the chips directly from their source rasters during training. If this can be done efficiently, this would remove the abovementioned issues. One thing to note is that it is still desirable to perform at least one offline step reducing the raster to the desired input resolution. This is necessary to avoid a computationally costly resizing operation during training that is unavoidable if the input raster images are of a different resolution than the desired input resolution.

4.7.1 Methods, Results, and Discussion

A generative image pipeline that produces image crops and labels directly from raster source files was developed and tested. Aside from the image loading, all settings were baseline. The direct loading trained at the same speed as the old data pathway after caching, but without the caching or preprocessing step being needed. It is not surprising that the time per epoch is the same – in the original pathway, after chips are loaded and resized they are cached in a serialized file for rapid loading so as to not repeatedly resize the same images.

However, this cache file is quite large – about 50 gigabytes – and takes time in the first epoch to create. The new pathway therefore saves time and storage space (the total size of the low res rasters used for input is only 3 gigabytes). In addition, the new pathway is much easier to alter without time consuming offline preprocessing.

References

- [1] L. Beyer, X. Zhai, and A. Kolesnikov. “Better plain ViT baselines for ImageNet-1k”. In: *arXiv preprint arXiv:2205.01580* (2022).
- [2] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, et al. “Language models are few-shot learners”. In: *Advances in neural information processing systems* 33 (2020), pp. 1877–1901.
- [3] F. Chollet. “Xception: Deep learning with depthwise separable convolutions”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 1251–1258.
- [4] P. Cui, W. Hu, and J. Zhu. “Calibrated reliable regression using maximum mean discrepancy”. In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 17164–17175.
- [5] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. “Bert: Pre-training of deep bidirectional transformers for language understanding”. In: *arXiv preprint arXiv:1810.04805* (2018).
- [6] T. DeVries and G. W. Taylor. “Improved regularization of convolutional neural networks with cutout”. In: *arXiv preprint arXiv:1708.04552* (2017).
- [7] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, et al. “An image is worth 16x16 words: Transformers for image recognition at scale”. In: *arXiv preprint arXiv:2010.11929* (2020).
- [8] Y. Gal and Z. Ghahramani. “Dropout as a bayesian approximation: Representing model uncertainty in deep learning”. In: *international conference on machine learning*. PMLR. 2016, pp. 1050–1059.
- [9] D. Hendrycks and K. Gimpel. “Gaussian error linear units (gelus)”. In: *arXiv preprint arXiv:1606.08415* (2016).
- [10] J. Hoffmann, S. Borgeaud, A. Mensch, E. Buchatskaya, T. Cai, E. Rutherford, D. d. L. Casas, L. A. Hendricks, J. Welbl, A. Clark, T. Hennigan, E. Noland, K. Millican, G. v. d. Driessche, B. Damoc, A. Guy, S. Osindero, K. Simonyan, E. Elsen, J. W. Rae, O. Vinyals, and L. Sifre. *Training Compute-Optimal Large Language Models*. 2022. DOI: 10.48550/ARXIV.2203.15556. URL: <https://arxiv.org/abs/2203.15556>.
- [11] O. Kesemen, B. Tiryaki, E. Özkul, and Ö. Tezel. “Determination of the Confidence Intervals for Multimodal Probability Density Functions”. In: *Gazi University Journal of Science* 31.1 (2018), pp. 310–326.
- [12] R. Liu, J. Lehman, P. Molino, F. Petroski Such, E. Frank, A. Sergeev, and J. Yosinski. “An intriguing failing of convolutional neural networks and the coordconv solution”. In: *Advances in neural information processing systems* 31 (2018).
- [13] J. Long, E. Shelhamer, and T. Darrell. “Fully convolutional networks for semantic segmentation”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 3431–3440.
- [14] I. Loshchilov and F. Hutter. “Decoupled weight decay regularization”. In: *arXiv preprint arXiv:1711.05101* (2017).
- [15] J. A. Nelder and R. Mead. “A simplex method for function minimization”. In: *The computer journal* 7.4 (1965), pp. 308–313.

- [16] G. Palmer, S. Du, A. Politowicz, J. P. Emory, X. Yang, A. Gautam, G. Gupta, Z. Li, R. Jacobs, and D. Morgan. “Calibration after bootstrap for accurate uncertainty quantification in regression models”. In: *npj Computational Materials* 8.1 (2022), pp. 1–9.
- [17] L. N. Smith. “A disciplined approach to neural network hyper-parameters: Part 1—learning rate, batch size, momentum, and weight decay”. In: *arXiv preprint arXiv:1803.09820* (2018).
- [18] M. Tan and Q. Le. “Efficientnetv2: Smaller models and faster training”. In: *International Conference on Machine Learning*. PMLR, 2021, pp. 10096–10106.
- [19] M. Tan and Q. V. Le. *EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks*. 2020. arXiv: 1905.11946 [cs.LG].
- [20] V. V. Tu. *Keras Implementation of Vision Transformer (An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale)*. 2020. URL: https://github.com/tuvovan/Vision_Transformer_Keras.

5 Deep Learning for Repairable System Reliability Prediction

5.1 Problem Introduction

DNNs have played a transformative role in the recent explosion of ML into nearly all spheres of industry and human endeavor. In diverse fields ranging from image generation to natural-language processing, DNNs have redefined what is possible with machine learning. Among these successes, it stands out that on structured or tabular data – such as one might find in any business database or spreadsheet – DNNs are typically not the ML method of choice. Certainly, one culprit is that non-DNN ML methods excel on tabular data. For practical reasons, most historical ML research was on structured data. Thus, there are many techniques specifically designed for tabular data that are not easily applicable to problems such as image processing – for example, tree-based methods. However, there is also a strong argument, as evidenced by a large and growing amount publications on this problem, that there remain many substantial methodological and technical advancements that could greatly improve the performance of DNNs on structured data. The possibilities for improvement are especially great on problems whose structure makes standard deep learning approaches ineffective, as is the case for the case study considered in this chapter.

The problem considered here consists of system attributes, operational data, and failure times from 8,232 oil and natural gas wells. The goal is to predict the frequency and (if possible) the specific timing of failures in the wells. Ideally, the prediction would apply both to novel wells not in the training data, as well as future times on wells already in the training set. This problem and dataset were presented previously by Liu and Pan [21], who developed an effective random forest based algorithm for predicting failure frequency. Each oil and gas well has an associated record of failure times, as well as a right censoring time past which data is not available. The data for each well consists of 8 static covariates, which do not vary with time but do vary by well, and 7 dynamic covariates which vary in time such as torque, load, and stress. The average observation period for a well is ~ 1823 time steps, and the average number of failures per well during the observation period is ~ 4.8 , ranging from 0 to 22. The overall data size is quite large, at 15,007,949 (of which 39,604, or .26%, are labeled as faults) time points each with 7 data points recorded variables recorded.

Failures in oil and gas wells are often costly and dangerous, and developing more precise means of predicting their occurrence has substantial value. However, in addition to being an important economic problem, this case study is a useful case study for methodological research on DNNs as it incorporates many challenging characteristics into a single problem. The problem is large, contains heterogeneous tabular data, is a time-series, has multiple subdivisions (wells), is right-censored per well, is in a domain where predictability is low, and is an extreme rare-event problem as well.

5.2 Deep learning approaches to tabular data literature

5.2.1 General Tabular Data Problems

Before moving into more specific problem domains, it is well worth looking at the state of the art on deep learning approaches to various tabular data challenges. For tabular data in general, a recent survey of the field by Borisov et al. [4] divides current approaches into three broad categories: data encoding approaches, architectural approaches, and regularization approaches. Data encoding approaches aim to improve model performance by transforming the format of input/output data. Architectural approaches aim to create neural network architectures that are specialized for handling tabular data. Regularization approaches aim to develop specialized regularization methods that better address the unique data challenges often present in tabular data (e.g. heterogeneous data). In practical applications, these approaches are not mutually exclusive and can be applied together - though the optimal combination of approaches for a given problem is something that must be experimentally determined.

Data Encoding

Data encoding is important to all neural networks, but the features of many tabular data sets provide a particular challenge and many possible choices. First, tabular data is typically heterogeneous, which means that the input and output features may vary widely in their type and statistical distribution. Continuous, integer-ordinal, and categorical data often coexist in the same tabular data set. Categorical data with a very large number of categories present special difficulties for neural networks which will be discussed shortly.

Second, neural networks typically only allow fixed-size inputs, which makes tabular data sets with varying feature amounts (common due to missing data) challenging to process. Third, it is often difficult to know in advance which variables are likely to have the most important interactions, which makes computation saving methods such as the convolutions used in CNNs difficult to find and design. For this reason, a large body of work has developed that aims to improve data encoding for deep learning on tabular data, of which only a few major developments and trends can be included here.

As mentioned, data with large numbers of categories can pose a problem for deep learning. One of the most common encoding technique for categories, "one-hot" encoding, does not scale well to variables with many possible categories. One-hot encoding creates a new feature column for each category, using a 1 or 0 to indicate whether the current item belongs to that category or not. In the case of many categories, this generates hundreds or even thousands of extra columns where almost every entry is 0. In addition to being extremely inefficient in terms of memory and computation, these large sparse inputs have been shown to severely inhibit the ability of deep learning models to learn effectively. A successful and increasingly popular method of dealing with these broad categorical variables are learned multi-dimensional embeddings. Originally conceived as a way to encode the large vocabularies of language models [3], this technique converts each category to an encoded form in a smaller, dense matrix. Crucially, the embedding itself is susceptible to learning, and so the model can learn more efficient and informative encoding over time. Dense embeddings are an essential part of the success of Transformer models on NLP tasks.

Specialized Architectures

With the enormous success of convolutional neural networks (CNNs) on image processing tasks, some authors have attempted to convert tabular data into images in order to use CNNs directly. CNNs exploit the inductive bias that pixels near each other in an image have more salient relationships than pixels that are further away. Thus, in order to usefully apply CNNs, the image conversion for tabular data must ensure that nearby pixels in the image are related to each other informationally. An informative example is the image generator for tabular data (IGTD) technique [30], which calculates distances between features

before converting them to pixels in an image to ensure that similar features are grouped together in the converted image. However, keep in mind that it is trivial to come up with distance measures which might make two features seem very closely linked when they are actually completely independent as far as influence on the problem task is concerned.

Finding effective neural network architectures for tabular data is an ongoing challenge. As seen above, one approach is to convert tabular data into forms that enable the use of effective architectures from other contexts, such as CNNs in image processing. In this case, the data is being made to match the architecture. The obvious alternative approach is to make the architecture match the data - and this is where specialized architectures for tabular data emerge. The large diversity and special challenges inherent in heterogeneous tabular data sets makes this a difficult task. One approach is to take successful non-deep methods and combine them with deep learning techniques. One of the more recent success stories is Neural Oblivious Decision Ensembles (NODE) [25] which combines a random decision forest algorithm with differentiable end-to-end deep learning. Among pure neural network approaches, Gorishny et al. [15] recently surveyed many different popular architectures on a variety of different tabular data sets and found that the two most effective approaches were a ResNet based model and a Transformer based model. The recent success of Transformer based models has also led to the development of several implementations for use on tabular data, such as TabNet [2].

Regularization

Finally, we come to regularization. In practice, many tabular data sets have properties (small size, input signatures that can easily uniquely identify records, lack of general augmentation strategies, imbalanced classes) that make them extremely vulnerable to overfitting. This makes regularization - always important - even more central to successfully implementing a deep learner on tabular data than in other applications. Kadra et al. [18] find that even multilayer perceptrons can achieve state-of-the-art performance on a wide variety of tabular data tasks when properly regularized by a "cocktail" of 13 different regularization techniques. However, it should be noted that not all regularization techniques can be applied to all problem types, and that most have at least one tunable parameter which

adds to the already large burden of hyperparameter tuning.

5.2.2 Deep Learning On Time-Series Data

Time-series data is a commonly encountered subtype of tabular data that presents unique deep learning challenges. The first challenge is how to handle input given that time series extend over time yet neural networks only admit a fixed size input. The simplest approach is to select some window size of time units and use that number of time units as a block of input. However, such an approach can never incorporate any information beyond its input window (consider, however, that Transformers achieve state-of-the-art performance on many sequence tasks yet use a window approach). Increasing the size of the window can provide more information, but it also increases the size of the network, and allows for more opportunities to overfit. One way to deal with this issue is to use stateful recurrent neural networks that can store information over long periods of time. The canonical example is the Long Short Term Memory network [16], which contains trainable memory and forgetting functions. LSTMs and other stateful approaches can be very effective, but they are difficult to train due to their strict requirements on data presentation order. Pertaining to the case study data set, the presence of 8,232 different subsequences would require stateful approaches to clear their memory many times per epoch, as the presentation order of the wells themselves should not be allowed to affect the results.

An alternative to both window and stateful approaches is to encode portions or even the entirety of a time series into some other format, such as an image. Since trends in time-series data are often visually distinguishable by human inspection, a deep learning image processing algorithm might also be able to perform this task. The intuition behind convolutions, that nearby pixels are generally more related to each other than further pixels, also applies to temporal adjacency in time-series data, making the direct use of CNNs on time-series data a natural extension. Convolutional kernels can be applied across multivariate time-series with little modification, using filters sizes that extend over some number of rows (time steps), as in [9]. Alternatively, standard 2-dimensional kernel sizes can be used if the data is encoded into an image format that converts temporal adjacency into spatial adjacency in the image. See [12] for a recent deep learning application comparing six different time-

series to image encodings. For the case study data set, the division of the time series into 8,232 subsequences provides natural breakpoints for a per well image encoding scheme, however the different sizes of the subsequences require special handling to ensure consistent input sizes.

Perhaps the largest challenge for neural networks on time-series data is extrapolation. Neural networks are notoriously poor at extrapolating beyond their training domain. However, the most commonly desired task with time-series data is to predict the future. As an example, consider that if absolute time is included as an explicit variable and encoded naively, this can lead to bad results on test data because the time inputs for test data will be outside the training set domain. Adding to this, most common neural network architectures do not easily encode cyclical features and thus will only fit cyclical curves within their training domain. Some ways to mitigate this concern include special encoding rules for time, from the simple (e.g. encode time as time-to-next-input, explicitly variablize relevant time periods such as weekends, night/day, seasons) to the complex (e.g. dense joint time-event embeddings [20]). Other ways involve modifying the network architecture to make encoding of cyclical functions natural, such as by using sinusoidal activation functions [13].

5.2.3 Deep Learning Approaches to Rare-Event Problems

Rare-event problems are a particular challenge for supervised data. The first hurdle is simply to ensure there are enough unique examples of the rare event in the training set to make supervised learning feasible. In the case study problem considered here, although failures make up only $\sim .26\%$ of the total data, the large overall data size means that there are 39,582 unique examples of failures available to train on. Next, even with sufficient samples, the highly imbalanced nature of the data will lead networks to ignore the rare event. This is problematic when failure to predict the fault (false negative) is more costly than erroneously predicting a fault (false positive). A number of different approaches have been suggested and developed to attempt to rectify this issue. The core of most approaches involve either modifying losses (class weighting), or over/undersampling. An alternative line of thought, however, is that class rebalancing is not necessary. Instead, decision-threshold-invariant metrics should be used which are not sensitive to class imbalance.

Class Weighting

Class weighting simply increases the gradient for the underrepresented class. This works well for mildly imbalanced data, but poorly for rarer events. In extremely unbalanced scenarios, the gradient adjustment necessary to create class parity over an epoch is very large. Consequently, the gradient descent steps taken when presenting the rare class become extremely large. Because the gradient constantly changes in the loss surface, gradient descent relies on taking relatively small steps where the calculated gradient is still (usually) a valid direction of improvement. Taking very large steps will easily take the model outside of the region where the calculated gradient of improvement was valid, leading to unstable and nonimproving models.

Over/undersampling

An alternative approach is to increase the frequency at which examples of the rare class are presented to the model, or reduce the frequency with which the common class is presented (oversampling and undersampling respectively). This works better in cases of extreme class imbalance due to avoiding the pitfall of overly large gradient steps. However, in cases of extreme class imbalance, simple oversampling means that the same rare class training examples will be presented to the model many times, creating an easy opportunity for overfitting. Simple undersampling throws out a significant quantity of data. More sophisticated oversampling techniques such as SMOTE [6] create novel minority class samples by making partial interpolations of similar class members. This works well so long as the assumption that small interpolated perturbations of similar samples still maintain the identifying data characteristics of the minority class. In general, it is not possible to know whether this assumption holds a priori, but the technique often shows good results in practice (depending on the metric), and has been successfully applied in deep learning contexts (see [29] for an example). However, as mentioned, when assessed using threshold-invariant metrics, class rebalancing techniques often provides no benefit and can even be detrimental. Thus they should be avoided unless absolutely necessary (for example, when using techniques that only produce a decision and not a probability or continuous score).

Anomaly Detection

Anomaly detection is an important subfield of rare-event problems. Within the literature, supervised deep anomaly detection methods are directly comparable to rare-event deep learning methods already discussed. The major differences between rare-event problems in general and anomaly detection problems specifically tend to be the level of labeled anomalous event data available, the breadth of anomalous events to be detected, and the desire to be able to identify novel anomalous events outside of the training data. In anomaly detection problems, it is common for the level of labeled anomalous data to be too small to effectively train in a supervised approach, for the goal to be to catch many types of anomalies, and for the model to have some capacity to flag anomalies even that it has never been trained on (which requires unsupervised learning). As an example of a common unsupervised deep approach, consider an autoencoder that is trained solely on non-anomalous examples. A poor reconstruction score, then, may indicate an anomaly in the input. See [1] for an example of autoencoder based anomaly detection model, and [24] for an up-to-date survey of deep anomaly detection methods.

5.2.4 Non-Deep Approaches

As mentioned, a major reason that deep learning approaches often do not exceed non-deep approaches on tabular data is the great strength of the non-deep methods in this domain. The current non-deep state-of-the-art for problems such as the case study, gradient boosted decision tree techniques such as XGBoost ([7]), are extremely powerful. They continue to see more research and development of specialized and improved methods for specific problem types such as CatBoost for categorical data [11]. This work would not exist if not for the work of Liu and Pan [21] who proposed a specialized and effective random forest algorithm for this large and challenging data set. Even if deep learning eventually supplants non-deep techniques on large-scale tabular data problems, non-deep methods will still have many advantages due to having a lighter computational footprint and much smaller data and development/tuning requirements. Because of these advantages, it will always be good practice to try non-deep methods on tabular data problems first before investing time into developing a deep learning model.

5.2.5 Deep Learning Approaches to Repairable Systems Literature

Finally, we come to the most directly applicable body of literature: other applications of deep learning techniques to repairable system problems. Although a relatively specialized subfield, the many challenges that these types of problems pose to standard approaches has led to great diversity of approaches in the literature. Yousefi, Tsianikas, and Coit[28] approached a multi-system maintenance model from an agentic standpoint, using reinforcement learning to learn an effective repair policy. Chen et al.[8] used a similar strategy on a multi-state repairable system problem. Nguyen, Khanh, and Medjaher [23] use an LSTM based network to predict failure times. However, they take the additional step of quantifying and modeling the costs of imperfect predictions, and what maintenance strategies should be taken in the face of uncertain information. Wang, Taylor, and Rees [27] performed a recent two-part review of the field and concluded that the most repairable systems data sets are too small for DL models to perform strongly, particularly as data is especially in demand due to imbalanced data where faults are rare. The case study data set, being very large relative to many data sets that have been so-far published in the literature, is an exciting opportunity to see if DL methods can overcome the difficulties of the problem domain given enough data. There is some precedence - NLP deep learning models showed rather stagnant performance for a long time. Then, they dramatically scaled up in capabilities with larger, more efficient models trained on more data. It is possible that we might see similar jumps in performance from other problem domains as we scale up data and model size/efficiency.

5.3 Methods Introduction

5.3.1 Feature/Label Considerations

Perhaps the most important consideration, aside from the gathering of data itself, when approaching a problem in machine learning is how that data will be defined and formatted into inputs and outputs. For some problems, such as image labeling, this is very simple as there are only a few logical arrangements. For problems such as the that described in this chapter, there may be many valid ways of organizing the data into inputs and outputs with advantages and disadvantages to each. The choice of output format also constrains and guides the choice of loss function, the choice of which can easily be the difference between

successful or unsuccessful applications.

In this problem, we wish to determine the frequency or time of system failures. The most direct format for our output then would be to simply categorize each time point as either failure or non-failure. If this could be done directly with high accuracy, we would certainly have achieved our goal. However, some thought reveals that this labeling scheme may be suboptimal from both an algorithmic and practical perspective. Consider that under such a scheme incorrectly predicting that a failure would happen 1 time step before a real failure actually occurred, and 1,000 time steps in advance of the nearest failure, are both equally wrong as measured by the loss function. Clearly, however, the temporal distance of a failure prediction to an actual failure will be of importance to owners and operators of repairable systems. Furthermore, since the problem is time-series data, we should expect temporal patterns in the data leading up to a failure. That is, we should expect data leading up to a failure event to be similar to each other, and different from data not leading up to a failure event, in some way that allows prediction of failures to be possible. If we don't believe this is true, then we don't believe that failure prediction from the data is possible in principle and should move on to another problem. All of this is to say that if we expect that the data characteristics of time leading up to a failure to differ from data characteristics in time not leading to a failure, the labeling scheme should reflect this.

However, even acknowledging this we have an infinite variety of differing possible labeling schemes. For example, we could use the time until next failure, or perhaps the time until nearest failure, as a label. However, this approach suffers from a different challenge. Previously, the problem was that the data labeling did not properly account for the expected presence of detectable (in principle) similarities near to failure events. Now, it fails to account for the expected similarity of data far from failure events. That is, we do not expect failures to be predictable an arbitrary amount of time in advance. Suppose for the sake of discussion that failures were only predictable n time units in advance. If this is the case, then the characteristics of the input data for any time point more than n units in advance of a failure should be the same (at least as far as predictive power for the time of the next failure is concerned). However, using a time-to-next-failure labeling scheme these statistically identical time points may have very different labels.

This brings us to the general principle that should inform any decision for formatting data, whether features or labels: data should reflect the functional relationship between inputs and outputs. Deep neural networks are fundamentally a method of functional approximation. They are themselves deterministic functions connecting inputs to outputs. What defines a functional relationship is that the same inputs should always have the same outputs. When we choose a labeling scheme, we should always try, to the best of our ability, to choose one that maps similar inputs to consistent outputs. If we fail to do this, we have given our network an impossible task: if, in the training data, the same input corresponds to two different outputs, no function can ever be found to map that relationship exactly.

A big problem for DNNs on tabular data is that we often don't know to what extent there even is a functional relationship between the input data and the desired labels. This is in contrast to problems in, for example, computer vision. Although we might not be sure of what it is, we can be confident that a function exists mapping images of dogs and cats to labels of dogs and cats. After all, we ourselves can perform that task. Coming back to our case study, we do not know in advance to what extent the task is even possible, and this adds considerable complexity to the problem of applying DNNs. For example, it could be the case that some failures are predictable from the data, and some failures are not. If this is true, the presence of the non-predictable failures in the same data class as the predictable failures is effectively noise in the input data that may interfere with learning to predict those failures that are predictable. But even if we knew this were true, how would we go about separating non-predictable failures from predictable failures before we had trained a predictive model?

In this work, both time-to-failure and distance-to-failure auxiliary labels are implemented and assessed.

5.3.2 Architecture

A CNN based deep learning method was designed and implemented. CNNs are a simple and effective architecture for any data with a spatial or temporal structure. In this case, 1d convolutions are used over the temporal dimension of the dynamic features (treating the dynamic features as a very tall, single channel image). Padding can be utilized to allow for inputs of variable length or forecasting. Since the most recent time points are expected to

be the most informative of the future, their order is reversed to keep them stationary in the input. This is so that they are in a constant location from sample to sample, allowing weights to train more effectively. Static covariates are input in a separate, non-convolutional branch, then concatenated to the CNN branch's output before a final MLP head that outputs the various losses. The target output label will be the next time step's dynamic covariates, as well as the direct failure label and the chosen auxiliary label. See Figure 5.1 for an overview of data organization.

Predicting the next time step, though not the primary goal, is generally useful for several reasons. Firstly, as shown in [26], auxiliary reconstructive losses can improve model performance on other labels by enforcing additional information to be carried through the network. Valuable information is thus potentially made available to the classifier. Additionally, the ability to predict the next time step may in and of itself produce value, if the predicted variables are of interest. Lastly, predicting the full next time step makes arbitrary forecasting possible, as the current prediction can be used as input for predicting the next. When using regularization techniques such as weight decay, or adaptive optimizers such as Adam, the weights of your model used for classification are in competition with the weights of the model used for other things. Thus, the label losses are weighted so that the classification loss is never overwhelmed by other losses.

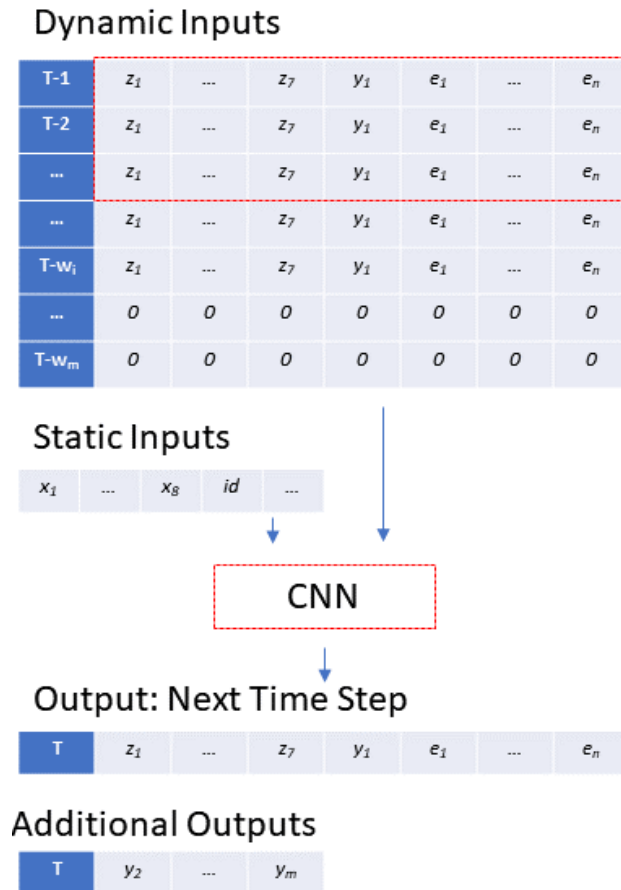


Figure 5.1: A diagram illustrating the basic organization of the approach. Dynamic covariates are input with a variable size w_i , up to a maximum window size w_m , where any extra space in the input is padded. The most recent timestep (T-1) is always presented first so as to keep it static in the computation graph. In addition to the dynamic covariates z , and the fault status y , the auxiliary labels are also used. The dynamic variables are processed by 1d convolutions, illustrated by the red dotted square. Static and dynamic inputs are combined, and the outputs consist of the predicted input for the next step, plus the labels and auxiliary labels.

5.3.3 Feature Embedding

Recently, Gorishny, Rubachev, and Babenko[14] showed that trainable embeddings of numerical features could have a substantial impact on performance. To see why numerical embedding can improve the model, consider it as akin to an additional preprocessing transformation. We observe that transforming one number into a mathematically related number, such as centering a variable around zero and normalizing its variance, can improve performance. But why not allow the model to learn its own data transformations? This is exactly what neural networks are always doing – transforming inputs into very large matrices using learned weights. The novelty is in the type of embedding, and in applying embedding immediately after input even if you’re using other architectures. For this project, the PLR – periodic, linear, relu – style of embedding introduced by Gorishny et al. is used. This was found to be the best overall performer in their study, and periodic activations are particular

appropriate for time-series data[13].

5.3.4 Regularization

Regularization is an extremely important aspect of neural networks in general and on tabular data especially. Neural networks are especially prone to overfitting on tabular data, leading to an entire body of research into regularization techniques for training neural networks on tabular data. For example, Kadra et al. [18] investigate a “cocktail” of 13 different regularization techniques, finding that a tuned, regularized multiplayer perceptron is capable of outperforming XGBoost. However, the reason regularization is so important to most tabular data problems is that, generally, tabular datasets are small. In this case, with over 15,000,000 samples, regularization is likely to be less of important, especially if the data has any random augmentations. Therefore, the model implemented here uses only a few different regularization techniques: a low level of dropout, model ensembling via moving average, and the weight decay of AdamW [22].

5.3.5 Measuring Model Performance

Binary classification is a very common problem type, so it is not surprising that many different metrics to assess classifiers under different criteria. For severely unbalanced datasets, however, many traditional metrics become misleading. The simplest example is the accuracy, where simply predicting the majority class all the time leads to a very high accuracy. If pure accuracy is all one is looking for, this is fine. But in most rare event problems the detection of the rare event is far more valuable – and missing it far more costly – than detecting true positive and finding false positives. To accommodate this fact, we can take any existing classifier that outputs probabilities and simply change the threshold at which we classify something in the positive class. But this will radically change most metrics, despite the fact that nothing has changed about the predictive model. What is required is a method to assess model performance without dependence on a particular choice of threshold – a threshold independent evaluation metric. One of the most widely used and best studied such metrics is the Area Under the Curve or AUC, as applied to the Receiver Operator Characteristic (ROC) curve or the Precision Recall (PR) curve. Briefly, the AUC

method integrates the chosen curve over all choices of threshold. Because it integrates over all threshold values, it is not dependent on any particular one. The ROC curve is the true positive rate over the false positive rate and has the property that it is insensitive to class imbalance. A random classifier achieves an ROC AUC score of .5, while a perfect classifier achieves a score of 1. The PR curve is, as the name suggests, a curve of precision over recall. Precision is the fraction of all positive predictions that are true, while recall, or sensitivity, is the fraction of all positive cases that are identified correctly. Although PR is threshold independent, it is not insensitive to class imbalance. The PR AUC score of a random classifier is the percentage of positive samples in the population, while the PR AUC score of a perfect classifier is also 1. Because the two scores measure different things, both are used here to assess model performance.

5.3.6 Class Balancing

This project does not use class rebalancing methods. In the planning phase for this project, class weights and undersampling/oversampling were suggested to address the heavy class imbalance of the data set. These methods are simple to implement and can give apparently great performance improvements on many metrics that are sensitive to data skew. They are correspondingly extremely popular. However, as discussed above, all of these gains can also be had by simply setting an appropriate decision threshold, without needing to change the model at all. The only time these methods produce true value is if you are bound to use the decisions output by a model directly with no choice of threshold – which can be the case for certain types of classifiers that only output a decision and not a score. However, in any case where you can simply choose the threshold, weight scaling is very much equivalent to simply choosing a different threshold. And when using a metric that is insensitive to skew, as suggested by Jeni et al.[17], it is quickly found in practice that class rebalancing techniques provide little benefit or even some detriment. Consider that none of these techniques add any information to the model. Undersampling even removes data, and oversampling/weight scaling distort true and useful information on the base rates of classes, which can lower PR score. To the extent that class imbalance is a true fact about the world, it is information the model should actually use. Thus, despite the large body of literature on them and their

frequent usage, the vast majority of class imbalance mitigation techniques may be worse than useless. Class imbalance is only a problem when it leads to an insufficient amount of samples of your minority class, so that generalization is impossible. The solution is getting more data, not trying to distort existing data. There is one exception to the rule, which are those methods such as SMOTE (Synthetic Minority Oversampling TEchnique)[5] that can add additional information to the model. SMOTE produces its samples through combinations/interpolations of existing minority class samples. But the value SMOTE brings, when it brings value, comes from data augmentation, not from correcting the class imbalance. And in fact, when used as a pure data augmentation technique, SMOTE can just as easily be applied to the majority class as well. It will actually work better because the larger number of majority class members means that nearest neighbors are usually closer together, and interpolations are more likely to produce in-distribution synthetic members. But even SMOTE, when measured using threshold insensitive metrics, rarely significantly improves models. This is because if the assumptions behind SMOTE are false (that is, if the chosen interpolations/combinations of samples create out-of-distribution examples), then SMOTE makes the model worse; and if the assumptions are true, then interpolation works, and an effective classifier will discover this and already score points in the interpolated region as the correct class when the threshold is set correctly. In other words, by design SMOTE produces the most likely to be in-distribution examples – these are safely inside regions of the input space that are already correctly classified. SMOTE rarely produces the sorts of close-to-the-boundary examples that actually help clarify decision boundaries, and by flooding the input with safe picks SMOTE may actually encourage overfitting, pushing the decision boundary away from the true manifold and worsening generalization outside of the training set. As a result of these issues and many pilot runs (data not shown) where various imbalance correction techniques did not improve the model as measured by threshold-invariant metrics, the final model does not use any kind of over/undersampling or class weights.

5.4 Methods Specification

Several deep learning binary classification models were trained on the data. Labels were 1 for failures and 0 for non-failures. Samples were prepared via the sliding window

method, with the chosen window size being seven based on pilot runs. The time gap for prediction was 0 (that is, we attempt to predict the label of the next point with no other points in between). None values (only present in the static covariates at low rates) were replaced with . For the marked variable input experiment, the input had varied forecast from 0 to 4 timesteps and varied sequence input length from 1 to 20, zero padding used as described above to fill the sequence. The dataset was split 80-20 into train-test datasets either according to time or to sites. The model also was trained using auxiliary labels, which were either time-to-failure or distance-to-failure. Auxiliary labels were scaled uniformly to be between 0 and 1, and also inverted so that failure is the positive class. Both distance and time-to auxiliary labels were capped at a maximum size – this maximum size was also used to fill any time points where calculation of the true auxiliary label was impossible (e.g. at sites with no failures, time-to-failure and distance-to-failure are ill-defined). In addition, model also predicts the next time point features for all dynamic features. Losses used were mean squared error for all outputs except the labels, which use binary crossentropy. The auxiliary label and label losses are further scaled relative to the other losses. The AUC of the ROC and PR curves are used as metrics for both types of labels.

The deep learning architecture utilized the periodic-linear-relu embedding as introduced by Gorishny et al. [14]. The architecture was as follows: on the dynamic inputs branch, periodic-linear-relu embedding on the inputs, followed by 3-layer 1-d convolutional blocks optionally ending with pooling (final model had no pooling). Padding was employed to maintain input size. The static inputs were concatenated to the output of the convolutional block before being flattened. A final 2-layer MLP block follows before the output. Activation of all dense and convolutional layers except the outputs was the Gelu activation function. The model has 351,318 trainable parameters. Hyperparameters were determined using hyperband[19] search (factor 6, max epochs 500, labels AUC as metric) in Keras tuner. The final values also received some final manual tuning after the tuner search. The model was trained using the AdamW optimizer, which also provides weight decay. The model also employed model averaging using the moving average with .99 decay. Batch size was 2048. Models were trained to convergence (4-6 epochs). Additionally, XGBoost[7] models were trained on the data for comparison. Due to computational constraints, only a limited

amount of manual tuning was performed on the XGBoost model, the parameters of the best performing model were as follows: learning_rate=.01; n_estimators=5,000; max_depth=4; min_child_weight=6; gamma=0; subsample=.8; colsample_bytree=.8; reg_alpha=.005; ¹.

Hyperparameter	Possible Values Searched	Final Model
Linear Embed Dims	1 to 8	3
Conv Layer Filters	8 to 256, step 16	152
Conv Filter Size	1 to 7, 3	160
Learning Rate	.01 or .001	.001
Number Conv Blocks	1 to 2	1
Conv Pooling	Max, Avg, or None	None
MLP Hidden Units	128 to 256, step 16	160
Label Loss Scale	25 to 85, step 30	250
Aux Loss Scale	.1 to 1	4
Weight Decay	3e-9 to 3e-3, log sampling	.0
PLR Periodic Dim	1 to 24	11
Embedding Type	PLR, LR, None	PLR
Dropout	.1 to .5	.1
PLR Sigma	.1 to 10, log sampling	27616

Table 5.1: Table showing search space and results of hyperparameter tuning. Some final values fall outside the search range due to manual tuning afterwards. *Auxiliary label scaling was implemented after the search, requiring this to be changed manually.

5.5 Results

See Table 5.2 for a summary of the results of the model under different data configurations. Table 5.3 further shows the AUCs for the auxiliary labels - note that these are calculated using the binary labels as the true values, but the auxiliary label predictions as the prediction. See Table 5.4 for results of additional models and experiments. The static only model has only the static covariates as input, while the dynamic only model has only the dynamic covariates. Figure ?? shows the representative ROC and PR curves for some of the models - the curves of models not shown with similar ROC and PR AUC values have similar shapes.

Aux Type	Data Split	Max Aux Value	ROC AUC	PR
Time-to-failure	Time	20	.5784	.0036
Distance-to-failure	Time	20	.5795	0.0036
Distance-to-failure	Site	20	.5720	.0034
Distance-to-failure	Time	10	.5792	.0036

Table 5.2: The results of different data configurations for the deep model.

¹For any parameter not noted, the default value was used, see <https://xgboost.readthedocs.io/en/stable/parameter.html> for a full list of parameters. Accessed 7/19/2022.

Aux Type	Data Split	MaxAux Value	Aux ROC AUC	Aux PR AUC
Time-to-failure	Time	20	.5476	.0032
Distance-to-failure	Time	20	.5722	.0035

Table 5.3: Table showing the distance between the results of the two auxiliary labeling scheme assessed using the auxiliary labels as the classifier score.

Experiment	Aux Type	Data Split	Max Aux Value	ROC AUC	PR
Random Seq/Forecast	Distance	Time	20	.5700	.0033
Static Cov. Only	Distance	Time	20	.5801	.0036
Dynamic Cov. Only	Distance	Time	20	.4981	.0026
XGBoost Default	NA	Time	NA	.5595	.0033
XGBoost Tuned	NA	Time	NA	.5727	.0035

Table 5.4: Table of results for additional models and configurations.

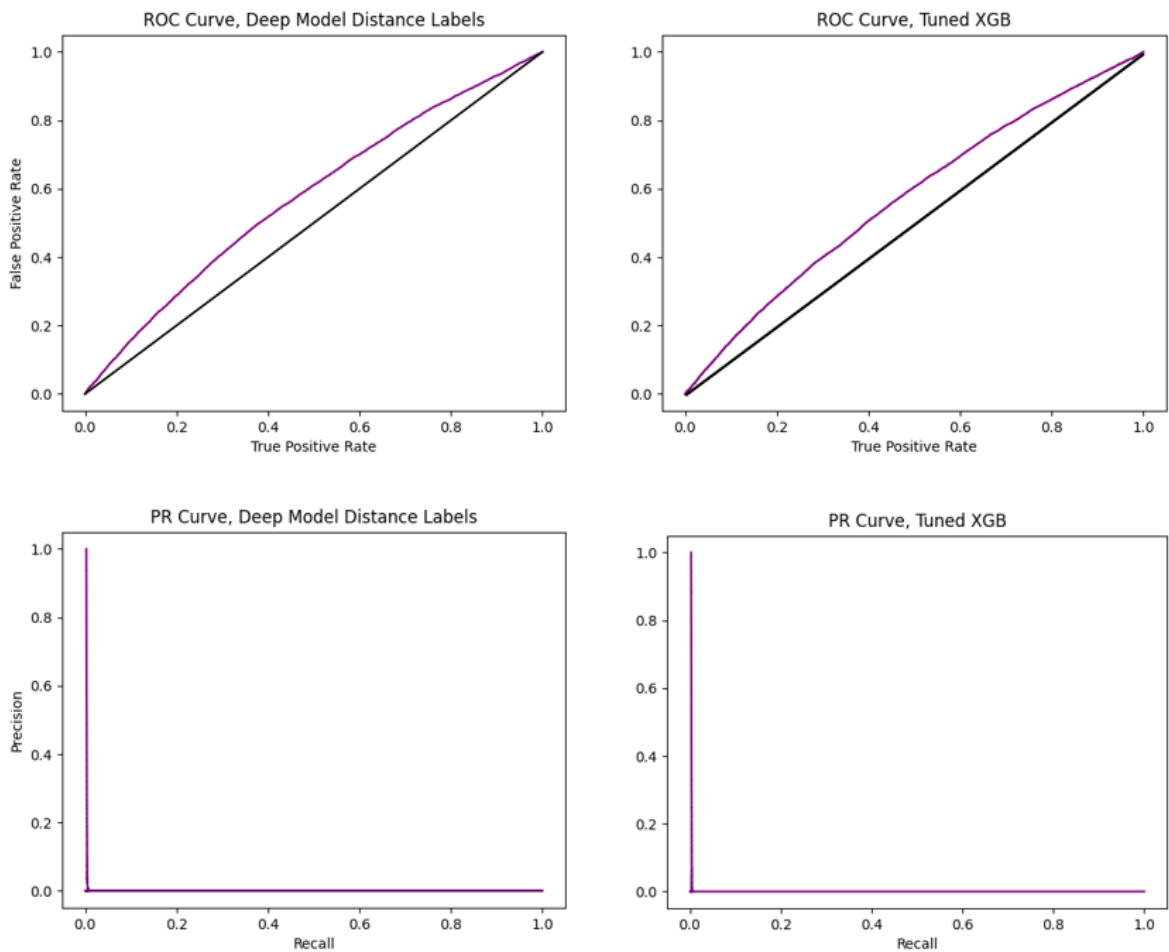


Figure 5.2: A figure displaying representative curves for the models. The black line represents chance performance.

5.6 Discussion

5.6.1 Model Performance

While not good in absolute terms, the final deep learning model achieved results that were noticeably above a random model (on PR AUC a random classifier scores .0026). This

is a good result when considering how challenging the dataset is expected to be. This is a problem and dataset where the predictive power of the features on the labels is unknown. Therefore, any result that is better than random chance is a potentially valuable finding, indicating that the features do, in fact, have at least some predictive value. Frustratingly, for datasets like this it is impossible to know just how much predictive power the features have until a model is made. Achieving a given level of predictive power in a model shows that the features have at least that much predictive value, but it is not possible to definitively say how much more accurate a model is possible. Thus, the model produced here could be very close to the best possible model for the given features, or very far. But, since machine failure is a highly complex stochastic process with many factors leading to it, it seems unlikely that input data that consists of only 7 dynamic process variables and 8 static site variables could pin it down with high precision (and high recall²).

XGBoost achieved comparable ROC and PR AUC scores to the deep learning model. Of course, the results shown can not definitely show that XGBoost, much less gradient boosted tree methods in general, can not do better than the deep model. Although XGBoost usually compares favorably to deep learning models in terms of speed of training, in this case the opposite was true due to the massive 15,000,000 sample dataset. XGBoost trains quite a bit more slowly than the deep learning model - which converges in just a few epochs - making tuning extremely costly in terms of time - the best performing tuned model took 12 hours to train. Because XGBoost was not the focus of this project, it was decided not to spend much time tuning it.

5.6.2 Data Troubles - Is the problem possible?

Performance was largely unchanged whether the training and test data was split by site or by time. Theoretically, it would be expected that site based splitting would be more difficult than time based splitting, as the network would be unable to learn patterns specific to the sites in the validation set it had never encountered. The lack of difference could be explained in two ways. First sites could just be an unimportant factor, so no difference is expected. Second, they could be an important factor, but in a way that is

²I apologize for this joke.

well captured by the static covariates so that the model can learn to generalize to new sites. However, experimentally removing static covariates from the input showed a dramatic decrease in performance, suggesting their importance. Thus, it seems that while site is an important factor, its importance can be well-characterized by the static covariates. The drop in performance without the static covariates was nearly to chance, suggesting that much of the predictive power of the model comes from the static covariates which do not vary over time (e.g. sites vary in overall rate in a way determined by the static covariates, but the dynamic covariates over time are not very informative).

This is an extremely troubling finding - if it is the case that the dynamic covariates have no or little impact on the rate of failure, then the dataset is not very valuable as a test case. It turns the problem from a time series problem with $>15,000,000$ data points into a much less interesting per-site failure rate regression problem with only 8,232 examples.

The random forecast and sequence length model did not cause much performance loss despite the problem possibly becoming harder due to variable length prognostication and inputs. However, when we consider the finding above that the static covariates are providing most of the predictive value, it becomes unsurprising. After all, the static covariates are time insensitive, so if all the predictive power of the model is based on them it does not matter how many dynamic time steps are included or how many time steps in the future the model is predicting. Unfortunately, because it uses random augmentation of the data, offline preprocessing and caching was not nearly as applicable to this approach. As a result, it trained significantly more slowly than any other model (6 times), and so was not used for tuning or comparative experiments. Additionally, as is the case with the auxiliary labels, there isn't a clear equivalent when training a gradient boosting tree to training a deep learning model with random augmentations or regularization. It is impractical to try and generate every possible result of a random input transformation to use in a gradient boosted tree model, but also unclear how many would need to be used to have a fair comparison to a deep learning model trained with such data.

The design and tuning of the deep learning model was informative, but also tarnished by the issues with the dynamic covariates. The periodic-linear-relu embedding from Gorishny et al.[14] seemed effective, but as the embedding was only applied to the dynamic covariates,

which may be unimportant, it's possible that this improvement was an artifact of some kind. The best model was not very large nor very deep, perhaps due to the limited size of the input data (a window of 7 on 7 dynamic covariates, plus 8 static, for a total of 57 inputs; in contrast, the single channel 28x28 MNIST [10] images produce 784 inputs), or simply the limited predictive power of the features. Other factors, however, led to more surprising findings. Firstly, in contrast to Kadra et al.[18], high regularization ended up being unimportant if not detrimental to model performance - the best parameter for weight decay during tuning was so low that it was decided to discard it entirely. This contrasting finding is probably due to two factors. First, the very large size of the dataset relative to the parameter count. With 15 million inputs samples and only around 300,000 trainable parameters, it is difficult to overfit on specific samples. Second, the weak signal of the data and the rare event nature of failures. If the predictive power of the model comes largely through the time-insensitive static covariates, then the model must accumulate statistical information on the rates of failures given certain values of the covariates over the entire very large data set. With weight decay on every step, it becomes extremely difficult for the model to properly accrue accurate statistics over 15,000,000 data points as data representations are constantly being decayed. Although extensive regularization cocktails proved unnecessary, a small dropout value and moving weight averaging was still useful. However, the lack of regularization might have prevented beneficial output interactions from arising.

Auxiliary Label Discussion Barring perfect classification, the ideal form of a classifier should take the form of a well calibrated probability of failure at each time point. The ideal training data for such a classifier would be the true probability of failure at each point. However, since this is unknown, the next-best training data is the closest approximation that can be made. From this perspective, treating time points as if they have point probabilities of 1 or 0 as actual ground truth probabilities is extremely unrealistic. This is the inspiration for alternative labeling schemes. What we would like to do is replace the point labeling scheme with a different one that more closely approximates the true probabilities of failure at each time point. The challenge faced here is to try and estimate the probabilities of a nonhomogeneous Poisson process where the rate is a function not of time, but the observed

process data³.

In the very likely case where certain signifiers in the data make a failure more likely to occur, but not deterministically at a certain time, binary labels are unrealistic. That is to say, a failure happens at a certain specific time t . However, it was probably also very nearly as likely to have happened at time $t-1$ or $t+1$. If this is the case, it does not make sense to label $t-1$ and $t+1$ with the complete opposite class as t , considering that they have nearly the same chance of failure. This issue is exacerbated the more fine grained the time steps are. Consider the same exact log of a machine operating normally for 10 hours with one failure. If the log is recorded in hours, there is 1 failure for 9 non-failure time points. If the log is recorded in minutes, then there is 1 failure for 599 non-failure time points. One unfortunate aspect of this data set is that it is not known exactly on what time scale it was recorded.

What properties does the rate function have? One possibility is that the rate is mostly driven by wear and tear. In this model, the probability of failure rises over time in some manner until a failure happens. Then, maintenance of at least some parts occurs, and the machine is restarted. After restarting, the probability of failure is at some reduced level due to the maintenance. A good labeling scheme should be time to next failure as it has many of the same properties. It decreases (indicating rising probability of failure) as we approach a failure, then resets afterwards to a high level (decreased probability). The most significant part of this model is that the probability of failure over time is discontinuous. The time points after a failure label should have decreased probability of failure, and the time points before a failure label should have an increased probability of failure.

An alternative model is to think that the failure pdf is mostly defined by transient phases in operation or environment. That is, the true probability of failure is driven by things like the current speed we are running the drill, or perhaps the type of soil we are drilling through, or if it is a pump what part of the pumping cycle, or any other sorts of operational characteristics we can image. Or perhaps restarting a well is a high-stress operation for the equipment. The key difference from the above model is that under this sort of scheme the

³To further connect the idea the NHPP vs. binary classification: a binary labeling scheme is equivalent to asserting that the rate of the NHPP is 1 for the time step a failure occurs and 0 everywhere else. Of course this is not true which is exactly why we want better labels.

rate function over time mostly looks like a series of peaks and valleys, and restarting the machine after a failure does not reset the probability of failure to a lower level. Under this model, the time points immediately after a failure still likely have an elevated chance of failure. A good choice of labeling scheme, then, would be something like distance to failure. Under a distance to failure labeling scheme, points before and after failures are considered to have an increased probability of failure.

It seems likely that both of these models are at least partially true. However, despite this, no auxiliary labeling schemes performed better than binary labels. Given the results of the static covariate experiments, it seems likely that the dynamic covariates are actually uninformative about the rate. Thus, almost all of the above chance performance of the model is just from time-insensitive rate estimates arising from the static covariates. Since the entire point of the alternative labeling scheme was to more realistically model the relationship of the rate function and the dynamic covariates over time, if there is no relationship in the first place it is unsurprising that there is no improvement. However, the distance labels performed noticeably better than the time-to-failure labels. It's possible that this indicates that the rate of failure does not decrease after a failure, or it may be due to time-to-failure not being centered around the failure labels (introducing a bias to predict labels too early).

Throughout various configurations during the tuning process (data not shown), no evidence was seen of substantial interaction between the different prediction variables - that is, the inclusion of additional losses didn't seem to affect the final scores of the other losses at convergence either positively or negatively. This is in contrast to the finding of many other papers using auxiliary losses (see, for example, [26]). Ideally, the different outputs and losses should result in intertwining flows of information within the model that augment each other. In practice, this did not occur. Certainly, and especially given the finding with static covariates, a major possibility is that there may genuinely not be very many genuine relationships between the different inputs and outputs of the model. Another possibility is that this is due to the lack of any weight decay or penalty. Weight penalties/decay encourages weight sparsity, thereby driving the model to use the same weights for multiple purposes when possible, increasing interaction between different parts of the model. Since the final model had no weight decay, it has no incentive in early training to share weights

over different sections of the model. During early training, the different loss gradients likely segregate into disjoint model flows for the different output predictions. During the later phases of training, interactions would be unlikely to arise as the different loss prediction pathways are already well established and independent, so there is little gradient crossover.

5.6.3 Future Work

Unfortunately, as far as can be determined from the models attempted here, the dynamic covariates have very little bearing on failure rate. If this is true, there is not much more to be done with it. Despite having some favorable properties as an experimental dataset due to its large size, if there is no relationship between the dynamic variables and the labels there is little reason to continue to develop models based on this data.

Training and computation limits were also an issue with the random prognostication and sequence length input data. While this model did seem promising, in that it maintained relatively close performance to the fixed window models while enabling inputs of different sizes and predictions at various points in the future, it was much more time consuming to train due to the need to dynamically generate and modify input data. This is a very attractive property, and also a major advantage over many tree-based models, where such capability would require training different models for each input length and advance forecast time. Methodological improvements that make training on this type of input more efficient would enable better exploration of these types of models.

5.6.4 Conclusion

A deep learning approach to a difficult failure prediction problem on a real-world dataset was developed and applied. The final model matched or exceeded the performance of XGBoost, a powerful tree-based method often providing state-of-the-art performance for tabular data classification problems, in a small test. However, since tuning of both models was limited by computational capacity, further investigation is needed to better establish the relative performance of the different approaches.

The results of the developed models seem to suggest that the dynamic covariates are not very useful for predicting the failure rate, and that most of the predictive power came

from the static covariates. Unfortunately, this indicates that predicting the failure rates from the features of this dataset in a time-sensitive manner is fundamentally not an achievable goal.

5.6.5 Acknowledgements

This material is based upon work supported by the National Science Foundation under Award No. OIA-1946391.

Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

References

- [1] J. An and S. Cho. “Variational autoencoder based anomaly detection using reconstruction probability”. In: *Special Lecture on IE 2.1* (2015), pp. 1–18.
- [2] S. O. Arik and T. Pfister. “Tabnet: Attentive interpretable tabular learning”. In: *arXiv* (2020).
- [3] Y. Bengio, R. Ducharme, P. Vincent, and C. Jauvin. “A neural probabilistic language model”. In: *Journal of machine learning research* 3.Feb (2003), pp. 1137–1155.
- [4] V. Borisov, T. Leemann, K. Seßler, J. Haug, M. Pawelczyk, and G. Kasneci. “Deep neural networks and tabular data: A survey”. In: *arXiv preprint arXiv:2110.01889* (2021).
- [5] K. W. Bowyer, N. V. Chawla, L. O. Hall, and W. P. Kegelmeyer. “SMOTE: Synthetic Minority Over-sampling Technique”. In: *CoRR* abs/1106.1813 (2011). arXiv: 1106.1813. URL: <http://arxiv.org/abs/1106.1813>.
- [6] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer. “SMOTE: synthetic minority over-sampling technique”. In: *Journal of artificial intelligence research* 16 (2002), pp. 321–357.
- [7] T. Chen, T. He, M. Benesty, V. Khotilovich, Y. Tang, H. Cho, et al. “Xgboost: extreme gradient boosting”. In: *R package version 0.4-2* 1.4 (2015), pp. 1–4.
- [8] Y. Chen, Y. Liu, and T. Xiahou. “A Deep Reinforcement Learning Approach to Dynamic Loading Strategy of Repairable Multistate Systems”. In: *IEEE Transactions on Reliability* (2021).
- [9] Z. Cui, W. Chen, and Y. Chen. “Multi-scale convolutional neural networks for time series classification”. In: *arXiv preprint arXiv:1603.06995* (2016).
- [10] L. Deng. “The mnist database of handwritten digit images for machine learning research”. In: *IEEE Signal Processing Magazine* 29.6 (2012), pp. 141–142.
- [11] A. V. Dorogush, V. Ershov, and A. Gulin. “CatBoost: gradient boosting with categorical features support”. In: *arXiv preprint arXiv:1810.11363* (2018).
- [12] G. R. Garcia, G. Michau, M. Ducoffe, J. S. Gupta, and O. Fink. “Time series to images: Monitoring the condition of industrial assets with deep learning image processing algorithms”. In: *arXiv preprint arXiv:2005.07031* (2020).
- [13] L. B. Godfrey and M. S. Gashler. “Neural decomposition of time-series data for effective generalization”. In: *IEEE transactions on neural networks and learning systems* 29.7 (2017), pp. 2973–2985.

- [14] Y. Gorishniy, I. Rubachev, and A. Babenko. *On Embeddings for Numerical Features in Tabular Deep Learning*. 2022. DOI: 10.48550/ARXIV.2203.05556. URL: <https://arxiv.org/abs/2203.05556>.
- [15] Y. Gorishniy, I. Rubachev, V. Khruikov, and A. Babenko. “Revisiting Deep Learning Models for Tabular Data”. In: *arXiv preprint arXiv:2106.11959* (2021).
- [16] S. Hochreiter and J. Schmidhuber. “Long short-term memory”. In: *Neural computation* 9.8 (1997), pp. 1735–1780.
- [17] L. A. Jeni, J. F. Cohn, and F. De La Torre. “Facing Imbalanced Data—Recommendations for the Use of Performance Metrics”. In: *2013 Humaine Association Conference on Affective Computing and Intelligent Interaction*. 2013, pp. 245–251. DOI: 10.1109/ACII.2013.47.
- [18] A. Kadra, M. Lindauer, F. Hutter, and J. Grabocka. “Well-tuned Simple Nets Excel on Tabular Datasets”. In: *Thirty-Fifth Conference on Neural Information Processing Systems*. 2021.
- [19] L. Li, K. Jamieson, G. DeSalvo, A. Rostamizadeh, and A. Talwalkar. “Hyperband: A novel bandit-based approach to hyperparameter optimization”. In: *The Journal of Machine Learning Research* 18.1 (2017), pp. 6765–6816.
- [20] Y. Li, N. Du, and S. Bengio. “Time-dependent representation for neural event sequence prediction”. In: *arXiv preprint arXiv:1708.00065* (2017).
- [21] X. Liu and R. Pan. “Analysis of large heterogeneous repairable system reliability data with static system attributes and dynamic sensor measurement in big data environment”. In: *Technometrics* 62.2 (2020), pp. 206–222.
- [22] I. Loshchilov and F. Hutter. *Decoupled Weight Decay Regularization*. 2017. DOI: 10.48550/ARXIV.1711.05101. URL: <https://arxiv.org/abs/1711.05101>.
- [23] K. T. Nguyen and K. Medjaher. “A new dynamic predictive maintenance framework using deep learning for failure prognostics”. In: *Reliability Engineering & System Safety* 188 (2019), pp. 251–262.
- [24] G. Pang, C. Shen, L. Cao, and A. V. D. Hengel. “Deep learning for anomaly detection: A review”. In: *ACM Computing Surveys (CSUR)* 54.2 (2021), pp. 1–38.
- [25] S. Popov, S. Morozov, and A. Babenko. “Neural oblivious decision ensembles for deep learning on tabular data”. In: *arXiv preprint arXiv:1909.06312* (2019).
- [26] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. “Going deeper with convolutions”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 1–9.
- [27] W. Wang, J. Taylor, and R. J. Rees. “Recent Advancement of Deep Learning Applications to Machine Condition Monitoring Part 1: A Critical Review”. In: *Acoustics Australia* (2021), pp. 1–13.
- [28] N. Yousefi, S. Tsianikas, and D. W. Coit. “Dynamic maintenance model for a repairable multi-component system using deep reinforcement learning”. In: *Quality Engineering* (2021), pp. 1–20.
- [29] H. Zhang, L. Huang, C. Q. Wu, and Z. Li. “An effective convolutional neural network based on SMOTE and Gaussian mixture model for intrusion detection in imbalanced dataset”. In: *Computer Networks* 177 (2020), p. 107315.
- [30] Y. Zhu, T. Brettin, F. Xia, A. Partin, M. Shukla, H. Yoo, Y. A. Evrard, J. H. Doroshov, and R. L. Stevens. “Converting tabular data into images for deep learning with convolutional neural networks”. In: *Scientific reports* 11.1 (2021), pp. 1–11.

6 Conclusion

As our world becomes more and more data-driven, machine learning will continue to have a greater and greater role to play in our society. When it is successful, deep learning has the ability to produce powerful models from data without requiring any knowledge of what functions connect the data to the desired output. The extraordinary potential of this power is only beginning to be unleashed. Deep learning straddles many areas traditionally in the domain of industrial engineers and is applicable to a wide variety of industrial and systems engineering problem domains. In this dissertation, several different applications of deep learning applied to practical and theoretical industrial engineering problems are presented. First, in the domain of absolute visual geolocation from aerial photography, a convolutional neural network model is successfully developed and applied. Then, that model was refined and thoroughly investigated, ultimately yielding techniques and suggestions to improve its capacity, accuracy, calibration, and practicality. Lastly, a deep learning model was applied to a complex and extraordinarily challenging dataset matching the performance of state-of-the-art methods like XGBoost, and although it ultimately revealed that the dataset was not well suited to the task, this too is a finding. Above all, the process of developing and applying the techniques and methodologies described in these projects, even when they did not produce the desired result, has expanded my capacity as an industrial and systems engineer.