Fall 1-2021

Applications of Machine Learning to Facilitate Software Engineering and Scientific Computing

Natalie Best
Chapman University, best120@mail.chapman.edu

Applications of Machine Learning to Facilitate Software Engineering and Scientific Computing

A Dissertation by

Natalie Claire Best

Chapman University

Orange, CA

Schmid College of Science and Technology

Submitted in partial fulfillment of the requirements for the degree of

Doctor of Philosophy in Computational and Data Sciences

January 2021

Committee in charge

Erik Linstead, Ph.D., Chair

Elizabeth Stevens, Ph.D.

Ruben Ramirez-Padron, Ph.D

CHAPMAN UNIVERSITY
SCHMID COLLEGE OF SCIENCE AND TECHNOLOGY
Computational and Data Sciences

The dissertation of Natalie Claire Best is approved.

Erik Linstead Digitally signed by Erik Linstead Date: 2021.01.18 12:49:11 -08'00'

Erik Linstead, Ph.D., Chair

Stevens, Elizabeth Digitally signed by Stevens, Elizabeth Date: 2021.01.13 18:18:07 -07'00'

Elizabeth Stevens, Ph.D.

Ruben Ramirez-Padron, Ph.D.

Date approved: January 2021

Applications of Machine Learning to Facilitate Software Engineering and Scientific Computing

by Natalie Claire Best

ACKNOWLEDGMENTS

I would like to thank my advisor, **Dr. Erik Linstead**, for not only his continued support, patience, and guidance but also for welcoming me into MLAT all those years ago.

To **Dr. Ruben Ramirez-Padron**, I am incredibly grateful for the generous feedback and mentoring while a member of my committee and also during my internship at Disney.

I would also like to thank **Dr. Elizabeth Stevens** for her advice and help, especially during my time as a Graduate Teaching Assistant.

I wish to thank the friends and colleagues who supported me along the way. Especially my bestie, **Dr. Viseth Sean**, whose friendship and positivity got me through the toughest times.

My deepest gratitude and appreciation goes to **Jordan Ott**, for his constant love, inspiration, and encouragement to accomplish hard things.

LIST OF PUBLICATIONS

Jordan Ott, Mike Pritchard, Natalie Best, Erik Linstead, Milan Curcic, Pierre Baldi, "A Fortran-Keras Deep Learning Bridge for Scientific Computing", Scientific Programming, vol. 2020, Article ID 8888811, 13 pages, 2020. https://doi.org/10.1155/2020/8888811

Natalie Best, Jordan Ott, Erik Linstead. "Exploring the Efficacy of Transfer Learning in Mining Image-Based Software Artifacts". *Journal of Big Data*, vol. 7, no. 59, 2020

Jordan Ott, Abigail Atchison, Paul Harnack, Natalie Best, Haley Anderson, Cristiano Firmani, and Erik Linstead. "Learning lexical features of programming languages from imagery using convolutional neural networks." *2018 IEEE/ACM 26th International Conference on Program Comprehension (ICPC)*, pp. 336-3363. IEEE, 2018

Abigail Atchison, Christina Berardi, Natalie Best, Elizabeth Stevens, and Erik Linstead. "A time series analysis of TravisTorrent builds: to everything there is a season." *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*, pp. 463-466, 2017

Abigail Atchison, Haley Anderson, Christina Berardi, Natalie Best, Cristiano Firmani, Rene German, and Erik Linstead. 2018. "A topic analysis of the R programming language." *In Proceedings of the 40th International Conference on Software Engineering: Companion Proceeedings (ICSE '18)*. Association for Computing Machinery, New York, NY, USA, 183–184.

ABSTRACT

Applications of Machine Learning to Facilitate Software Engineering and Scientific Computing

by Natalie Claire Best

The use of machine learning has risen in recent years, though many areas remain unexplored due to lack of data or lack of computational tools. This dissertation explores machine learning approaches in case studies involving image classification and natural language processing. In addition, a software library in the form of two-way bridge connecting deep learning models in Keras with ones available in the Fortran programming language is also presented.

In Chapter 2, we explore the applicability of transfer learning utilizing models pre-trained on non-software engineering data applied to the problem of classifying software unified modeling language diagrams where data is scarce. Our experimental results show training reacts positively to transfer learning as related to sample size, even though the pre-trained model was not exposed to training instances from the software domain. We contrast the transferred network with other networks to show its advantage on different sized training sets.

Implementing artificial neural networks is commonly achieved via high-level programming languages like Python and easy-to-use deep learning libraries like Keras. These libraries come pre-loaded with a variety of network architectures, provide autodifferentiation, and support GPUs for fast and efficient computation. Many large-scale scientific computation projects are written in Fortran, making it difficult to integrate with modern deep learning methods. To alleviate this problem, we introduce a software library, the Fortran-Keras Bridge (FKB), that connects environments where deep learning resources are plentiful, with those where they are scarce. Chapter 3 describes several unique features offered by FKB, such as customizable layers, loss functions, and network ensembles.

In Chapter 4, Latent Dirichlet Allocation (LDA) is leveraged to analyze R and MATLAB source code from 10,051 R packages and 27,000 open source MATLAB modules in order to provide empirical insight on the topic space of scientific computing. This method is able to identify several generic programming concepts and, more importantly, concepts that are highly specific to scientific and high performance computing applications. We are also able to directly compare these topics using document entropy and topic uniformity scoring.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF LISTINGS

LIST OF ABBREVIATIONS

ANN Artificial Neural Network

API Application Programming Interface

CAM Class Activation Mapping

CNN Convolutional Neural Network

CRAN Comprehensive R Archive Network

CRM Cloud Resolving Model

DNN Deep Neural Network

FKB Fortran-Keras Bridge

GPU Graphics Processing Unit

HDF5 Hierarchical Data Format version 5

LDA Latent Dirichlet Allocation

MAE Mean Absolute Error

MATLAB Matrix Laboratory

MSE Mean Squared Error

NN Neural Network

PNG Portable Network Graphics

ReLU Rectified Linear Unit

RMSProp Root Mean Square Propogation

SGD Stochastic Gradient Descent

SPCAM3 Superparameterized Community Atmospheric Model version 3.0

SVM Support Vector Machine

UML Unified Modeling Language

VGG Visual Geometry Group

1 Introduction

This dissertation presents three studies I have worked on during my time at Chapman University and is structured as follows. In Chapter 1, I introduce several methods that are used throughout the subsequent chapters. Presented in Chapter 2, is a study that employs transfer learning to enable classification in a domain that lacks sufficient data to train a typical neural network from scratch. In Chapter 3, a Fortran deep learning library is proposed that allows users to implement deep neural networks in Fortran and access these networks in Python with all of the existing Keras capabilities. In Chapter 4, the source code of two scientific computing programming languages is explored with Latent Dirichlet Allocation (LDA).

1.1 Machine Learning

The field of machine learning contains a wide range of techniques and methods aimed at teaching computers a specific task. The models generated are trained and validated by data in the form of examples or past experiences. Machine learning algorithms are typically broken down into three main subgroups; supervised, unsupervised, and reinforcement learning.

Supervised learning algorithms are those for which the targets, whether categorical or continuous, are known when training a model. For example, in Chapter 2, the task of classifying UML diagrams is considered supervised due to the fact that we know at the outset which images are sequence diagrams and which are class diagrams. Using the example inputs and corresponding known targets, we can improve our model incrementally during training to help it learn a general model for the data. Depending on the type of supervised model, the evaluation metric used to determine success will differ. For classification tasks, accuracy is a common

metric, which is defined as the proportion of correct predictions out of all predictions. In addition, metrics such as precision and recall may also be useful tools by which to judge a model. For regression tasks Mean Squared Error (MSE) or Mean Absolute Error (MAE) are popular metrics to determine overall how accurately a model can predict targets.

Unsupervised learning algorithms do not have access to any ground truth and must find the latent structure within a feature space. Types of unsupervised tasks include clustering and dimensionality reduction. In Chapter 2, underlying themes are generated from programming language source code using LDA, a popular unsupervised algorithm for natural language processing. To train a model, the algorithm requires a corpus of text documents along with declaration of several hyperparameters. However, because there is no test set. The topics it produces are what we call latent variables, or variables that are not directly observed but can be inferred from the data. The success of unsupervised methods is more difficult to judge due to the lack of truth data.

1.1.1 Deep Learning

Within the field of machine learning, deep learning employs Artificial Neural Networks (ANNs), whose architectures are inspired by the biological neural networks within the brain. ANNs can be trained to perform human-like tasks including speech recognition, image classification, and natural language processing. As shown in Figure 1.1, ANNs are made up of layers of nodes that are typically fully connected in which a vector of data is fed through to produce some prediction. Each node in the network is connected to all nodes in the previous layer and the output of a node is the dot product of the incoming weights and input values. This dot product is run through an appropriate activation function and the result is sent on to the next layer. Non-linear relationships in the data can be modeled by using sigmoidal activation functions, like the logistic or hyperbolic tangent functions. Weights within the network are trained using gradient descent to minimize the error between output and truth data through the backpropagation process. This is possible due to the activation functions' continuous and differentiable nature.

As ANNs grow in size with multiple hidden layers, we refer to them as a Deep Neural Network (DNN). For example, in Chapter 2, a DNN is used that contains 21 layers of various types. Then in Chapter 3, we present a library to enable the use of DNNs models in Fortran along with a bridge to transfer those models between Fortran and Keras, a popular Python deep learning library.

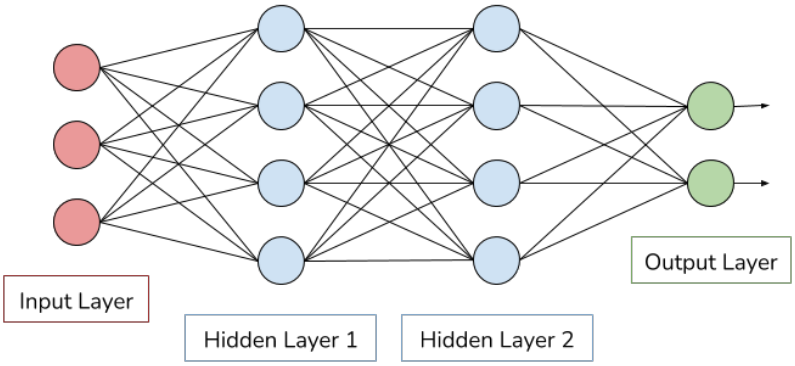


Figure 1.1: A simple Artificial Neural Network architecture with 2 hidden layers.

Convolutional Neural Networks

Convolutional Neural Networks are a type of deep learning architecture used in image analysis, due to the network's ability to preserve spatial information that would be lost in a fully-connected feedforward network. For example, if we had a 10 x 10 pixel image and were to send it through a fully-connected feedforward network, the image would need to be flattened into a 100 x 1 vector. In which case, the spatial relationship between pixels is erased.

The inspiration for Convolutional Neural Network (CNN) architecture is derived from experiments performed by Hubel and Wiesel on the cat visual cortex [1]. Convolutional networks incorporate a weight matrix, or kernel, that is convolved over the input image. Figure 1.2 shows an example image and kernel along with the resulting output. The green weights are slid over the blue image and the sum of element wise multiplication results in one element of the purple

feature. Typically, the weights of the kernel are shared over the entire image, although Free Convolutional Networks that do not use a shared weight paradigm have been explored [2]. In the case where weights are shared, the number of learnable parameters is drastically reduced. The default activation function used in conjunction with CNNs is the Rectified Linear Unit (ReLU) [3]. The sigmoidal and hyperbolic tangent functions are not typically used with deeper networks due to the vanishing gradient problem, which ReLU is able to combat. The output of ReLU is the maximum between the input, x_i, and 0: $y_i = max(0, x_i)$. The DNN used in Chapter 2 contains 16 convolutional layers with ReLU activations, out of a total 21 layers. This network is aptly named VGG-16. Training CNNs through backpropagation was first proposed by LeCunn et al.[4]. After the convolution layers there may be a number of full connected layers, allow us to condense the output into a more understandable and usable form.

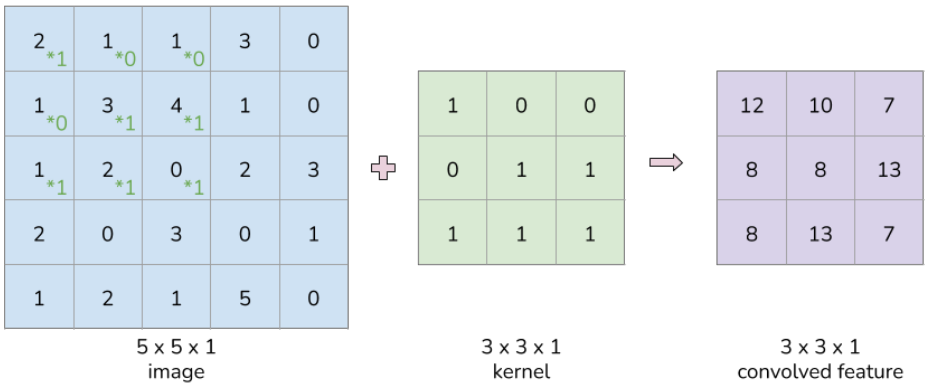


Figure 1.2: Example of convolution given an image and kernel.

Class Activation Mapping

Class Activation Mapping (CAM) is a visualization technique that allows us to investigate further what parts of an image a convolutional network uses to make its prediction [5]. Using the Keras Visualization Toolkit [6], we will produce CAM results in Chapter 2 of this paper. The result produced by CAM is a heat map indicating the features a network relies on most to make its prediction. The most heavily weighted areas of an image, are highlighted in red, while less important areas are blue.

In addition to showing the features of an image important to a network, CAM results also allow us to ensure those features make sense. For example, say we were to build a CNN to distinguish between images of cats and dogs. We could expect to see CAM highlighting features such as the face, ears, body, and tail of the animal. We would be concerned about our network and dataset if CAM results showed the backgrounds being the most important in its decision, e.g. grass for the dog photos and indoor furniture for cats.

Transfer Learning

Transfer learning is the process of taking a model trained for one task, where data is more readily available, and applying it to a new but similar task [7]. Traditionally, given two separate tasks, we would have to obtain two distinct training sets and build models for each task. Unfortunately, large amounts of data in every domain are not always available, and in a lot of cases are not always needed if two tasks are similar enough.

When considering how humans learn to do new tasks, they rarely have to start at the absolute beginning - tabula rasa - and typically are building off of similar previous experiences. If one tries to learn a new language, or how to play a new game, one draws on prior knowledge and adapts to complete the task at hand. This is core idea of transfer learning; to learn general features in one domain and apply those features to another, similar domain. Transfer learning can take a few different general forms depending on the source and target domains, as well as the source and target tasks [7]. Inductive transfer learning occurs when the source and target domains are the

same but the tasks to be performed differ. The inductive biases of the source task algorithm are used to help improve the target task algorithm. Unsupervised transfer learning is similar, in that the tasks differ, except that labeled data is unavailable in both the source and target domains. This type may include tasks such as clustering and dimensionality reduction. Transductive transfer learning occurs in scenarios where the source and target tasks are similar but the domains are different. In this situation, the source domain typically has a sufficient amount of training data while the target domain does not. This type of transfer learning will be used in Chapter 2.

In our case, we transfer general features learned when the VGG network has been trained on the ImageNet [8] dataset and fine tune it to the task of UML classification. We choose this classification task for our experiment for three reasons. First, the work in [9] used this same data to demonstrate the inability of deep networks such as VGG-16 to learn features when training samples are limited, requiring custom architectures to be built. Second, UML is sufficiently dissimilar to other objects found in ImageNet that we can be confident that pre-trained models will not have already learned features directly applicable to the classification task. Finally, the automated classification of software artifacts is an essential task when curating data on an Internet-scale as is typically the case in empirical software engineering studies. In our study, we will implement the same networks used by Ott et al. as baselines [9].

When applying transfer learning, a decision must be made to determine how much will be borrowed from the original algorithm. It is common practice to take an established architecture and freeze some amount of the original layers, while fine tuning the rest to the specific needs of a problem. As a result, only the unfrozen layers are trained - resulting in far fewer learnable parameters which decreases the size of the required labeled dataset for training. The amount frozen and fine-tuned is variable depending on the task at hand. We will explore two variations on the VGG-16 architecture, as well as a shallow CNN in Chapter 2. In one VGG network, we fine

tune all available weights and see poor accuracy when dealing with small training samples due to the large parameter space that must be learned. In the second, we freeze the majority of weights while fine tuning only the final layer and see accuracy near 90% even at very low numbers of training samples.

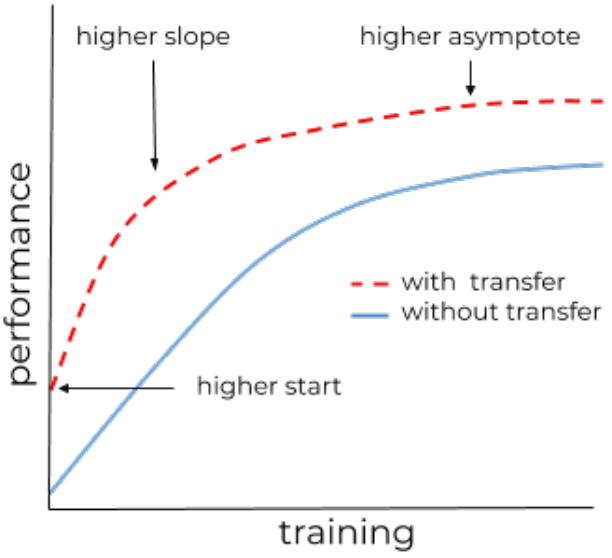


Figure 1.3: Three locations we expect to see improvement in model performance from a knowledge transfer.

In general, when implementing transfer learning, we must look in three areas for possible superiority over other networks, as shown in Figure 1.3 [10]. First, we may find a higher starting accuracy, at the beginning of training, before the model has been refined further. Second, we could see a steeper or faster rate of improvement of accuracy as training continues. Finally, we look for a higher asymptote, or greater accuracy toward the end of training. In our results, we find that the frozen VGG network exhibits higher accuracy in all three of these areas over the pre-trained VGG and a shallow CNN.

1.1.2 Topic Modeling

Topic modeling methods aid humans in organizing, understanding, searching, summarizing and searching massive collections of text documents. We are able to reveal the underlying themes that exist within the collection. From these general themes, individual documents can be tagged with those that apply. Although these tags may not be perfectly accurate, they enable us to organize, summarize, and then search the individual documents using those tags.

Latent Dirichlet Allocation is a statistical topic modeling algorithm capable of learning the underlying document-topic and topic-word distributions from a text corpus [11]. This unsupervised machine learning model is used in Chapter 4 to extract latent topics from the source code of two programming languages. LDA represents documents as bags of words, meaning that only the frequency of each word in the document, and no other language structure, needs to be taken into account. For a corpus, C, consisting of d documents and a total vocabulary size of v, we construct a document word matrix, DW, of size $d \times v$, with each row representing an individual document, and each column a unique word. Hence, the element $DW(i, j)$ consists of an integer denoting the number of times document i contains word j.

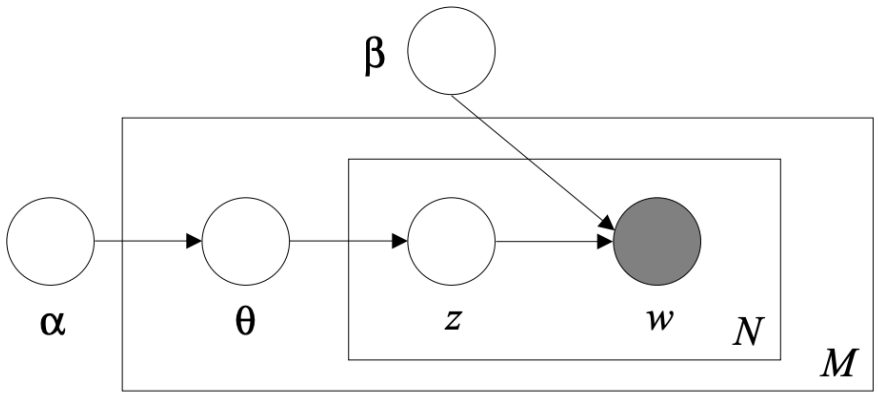


Figure 1.4: Graphical model representation of LDA.

As shown in Figure 1.4, the LDA model is represented as a probabilistic graphical model. In this diagram, the parameter α is the per-document topic distributions and parameter β is the per-topic word distribution. Further, θ represents the topic distribution for document *m*, φ is the word distribution for topic *k*, while z is the topic assignment for the *n*-th word in document *m*, and *w* is a word.

Given the document topic matrix, LDA learns the necessary posterior distributions to infer document-topic and topic-word mixtures, assuming an underlying multinomial model. The process is made Bayesian by adding Dirichlet priors, which are typically assumed to be symmetric for simplicity, though more recent works suggests asymmetric priors can be beneficial [12]. Training is accomplished via Gibbs sampling or variational Bayesian methods. The result is a full mixture model of document-topic and topic-word probabilities, which can form the basis for content-based classification or clustering models.

2 Exploring the Efficacy of Transfer Learning in Mining Image-Based Software Artifacts

2.1 Introduction

Despite the recent successes of deep architectures, such as convolutional neural networks, on software engineering data, the lack of sufficiently large training sets for some applications continues to be a substantial hurdle. This requirement has led researchers to label tens of thousands [13] and even millions of images [14] by hand. Recent work has shown that this precludes the use of many off-the-shelf convolutional neural network architectures, requiring empirical software engineering researchers to rely on custom (more compact) architectures [9]. Another possible solution, however, is to leverage transfer learning to deal with large parameter spaces. Through this process models learn in one domain - where data is plentiful - and *transfer* this knowledge to a domain where data is scarce.

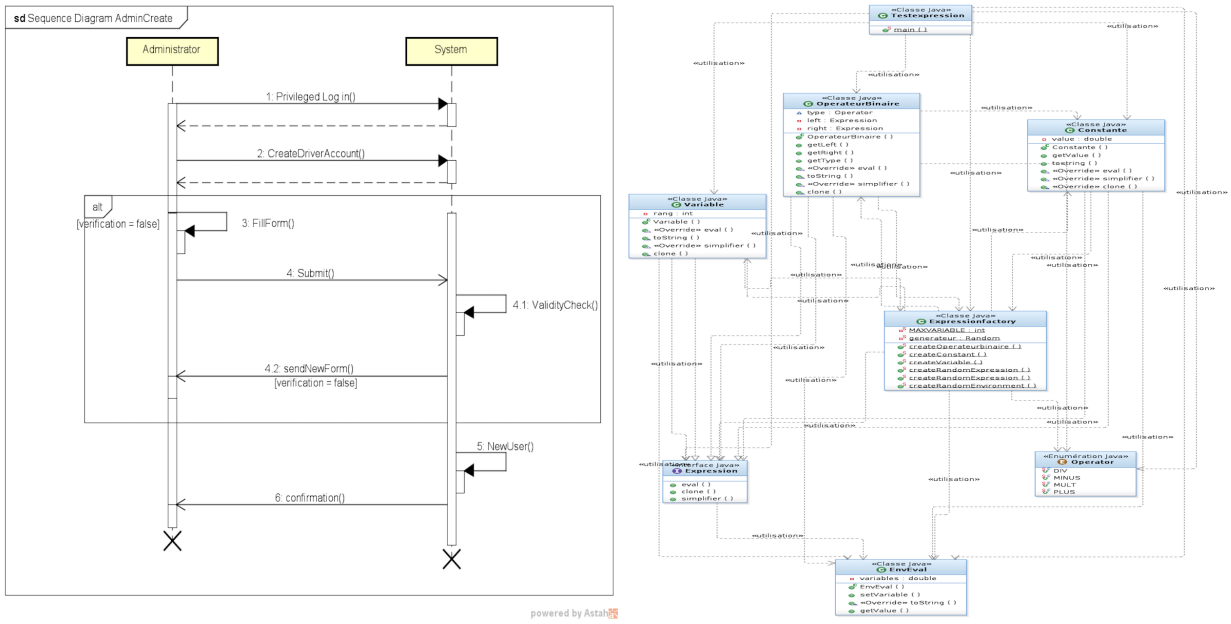
One significant limitation in deep learning is data dependence. As computational ability and available algorithms have improved significantly over the years, many deep learning techniques are still held back by the need for massive amounts of labeled truth data. As architectures increase in depth and number of parameters, the amount of data needed to train networks increases as well. When large datasets are not available, or are difficult to curate, researchers must turn to other methods in order to improve their models. Other possible solutions to small amounts of data have been investigated including low shot learning, meta-learning, and data augmentation [9]. Although, even with these other methods to combat small datasets, the bottleneck of large

parameter spaces and the computation time needed to train a deep neural network remains. As an example, the very deep convolutional networks developed by the Visual Geometry Group at the University of Oxford, take about 2-3 weeks to fully train the 130-140 million parameters in a network, depending on the architecture [15].

In this chapter, we explore transfer learning as a way to combat the issues related to limited data. Many publicly-available, state-of-the-art models already exist and have been trained on huge amounts of data including VGG [15], AlexNet [16], ResNet [17], and Inception[18]. These networks have repeatedly been applied to different tasks from which they were originally trained [13, 19, 20, 21, 22]. We will also apply an off-the-shelf architecture, fine tuning it to our task, to show the advantages of knowledge transfer when working with limited data in the software domain. We focus on the classification of UML diagrams into class and sequence diagrams from a publicly-available dataset [23]. This dataset has been previously leveraged to demonstrate barriers that arise when applying deep architectures with vast parameter spaces.

2.2 Data

From the Lindholmen Dataset [23], an initial corpus of 14,815 Portable Network Graphics (PNG) images of UML diagrams is obtained. That is then reduced to 13,359 images when only active UML diagrams are considered. Of the active diagrams, there were 11,319 Class Diagrams and 2040 Sequence Diagrams. Examples of these diagrams are shown in Figure 2.1. We resize all images to 250x250 pixels for uniformity. To resize a file, we sample the pixels depending on how large the original image was. For example, given a 1024x1024 pixel image, every 4th pixel would be used in the x and y direction, or 1024 // 250 = 4. This dataset was chosen for its small size and its relation to software repositories. The VGG-16 networks we include in our tests have been trained on the ImageNet dataset which includes over 1,000,000 natural images belonging to 1,000 categories. Although the natural images of ImageNet and UML diagrams exist in quite different domains, we still see improvement in classification when using knowledge transfer.

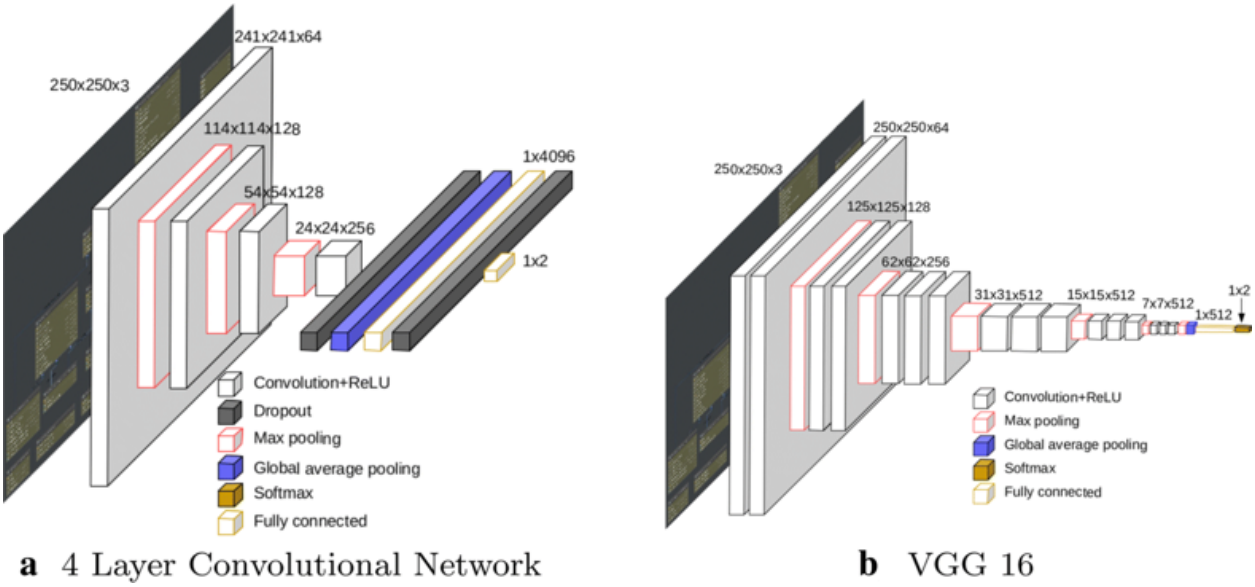


(a) (b)

Figure 2.1: One example of each type of diagram used in this study: (a) sequence diagram and (b) class diagram

2.3 Methods

In our experiments, we compare three CNN architectures on their classification ability of UML diagrams. First, we use a simple network with four convolutional layers, max pooling, dropout, and global average pooling layers followed by fully connected dense layers for classification. This network contains 2,260,000 trainable parameters. Two other networks explored are variations of the popular VGG network with sixteen convolutional layers modified to fit the size of our input data [15]. The first VGG we test starts with the original weights and we then allow all 14,715,000 trainable parameters to be updated as we train for our task. Conversely, in the second VGG, we freeze the majority of layers, and then modify and train only the last layer containing only 1,026 trainable parameters. The four layer CNN and VGG architectures are shown in Figure 2.2. All networks are implemented in Keras with a TensorFlow backend.



a 4 Layer Convolutional Network **b** VGG 16

Figure 2.2: Networks used a) The four convolutional layers, interspersed with max pooling for downsampling followed by dropout, max pooling, and fully connected layers for classification. b) Standard VGG network with sixteen convolutional layers.

These three models were trained as binary classifiers to differentiate UML diagrams as either sequence or class diagrams. To show the advantages of transfer learning, we incrementally increase the available training data in two tests. We begin with 50 samples of each class and increase by increments of 250 to 1800 samples. A second test to show the accuracies at very low samples is performed beginning with 5 samples and increasing by increments of 5 to 50 samples. Upon incrementing the sample size, each network is reset to the same original weights.

Each model was trained for a minimum of 5 epochs and stopped when the accuracy had not improved after a patience of 5 epochs. We implemented 5 fold cross validation for robustness. It is common practice to include a patience in order to control training time [24]. Therefore, when a model shows no signs of improving, and we have met an established minimum number of epochs, we are free to stop. For example, in our test of the 1800 diagram sample size, our frozen VGG network quickly reached an accuracy of around 93%, on each fold, after an average of only 15 epochs. Continuing to train would likely not improve our model by any significant amount and could even lead to overfitting.

The code and data to train all models, as well as the learned models themselves, are available publicly at: (removed for anonymity) We hope they, in turn, will be utilized for transfer learning in future deep learning applications on software data.

2.4 Results

Figure 2.3 shows the test accuracy achieved by each network from 50 to 1800 samples of each class, or 100 to 3600 total images respectively. Both the frozen VGG and 4 layer CNN are eventually able to classify the given diagrams with about 90% accuracy given a sufficient amount of samples. Although, we see a significant difference in the starting accuracies as well as faster convergence.

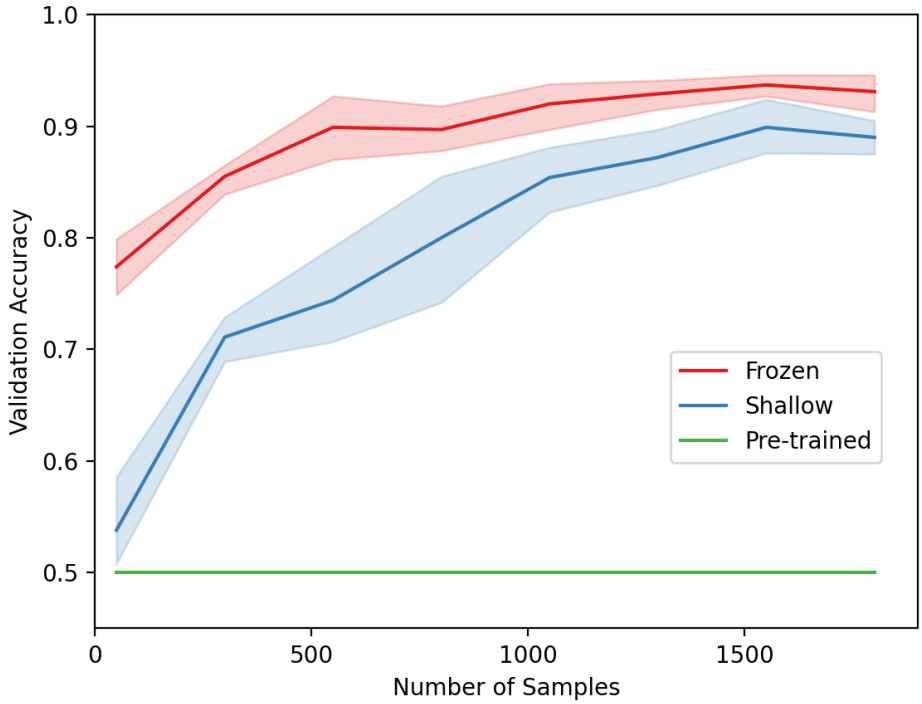


Figure 2.3: Accuracy achieved by each network at the corresponding sample sizes, from 50 to 1800 samples in each UML category. For robustness, 5 trials were run for all training samples tested. The color bands indicate the distribution of results from the 5 trials.

However, we are also interested in the best accuracy achievable with the least amount of data. The frozen VGG is able to classify with an about 80% accuracy after only 100 total training samples while the 4 layer CNN falls short at about 52% accuracy. As can be expected, the VGG that was left free to train the massive number of parameters within its network, also performs poorly, barely reaching 50% accuracy. In which case, it would be no better than simply flipping a fair coin to classify each diagram. The tiny amount of training data given to this network is, of course, nowhere near enough to train all 14 million parameters.

Figure 2.4 shows the training accuracy for all three networks when given 5 to 50 samples of each class, or 10 to 100 total images. We include this figure to demonstrate the superiority of the frozen VGG over both networks especially at very low samples. Even with only 10 total samples, the frozen VGG is able to classify the UML diagrams with an average 73% accuracy, compared to an accuracy of only 50% for both other networks.

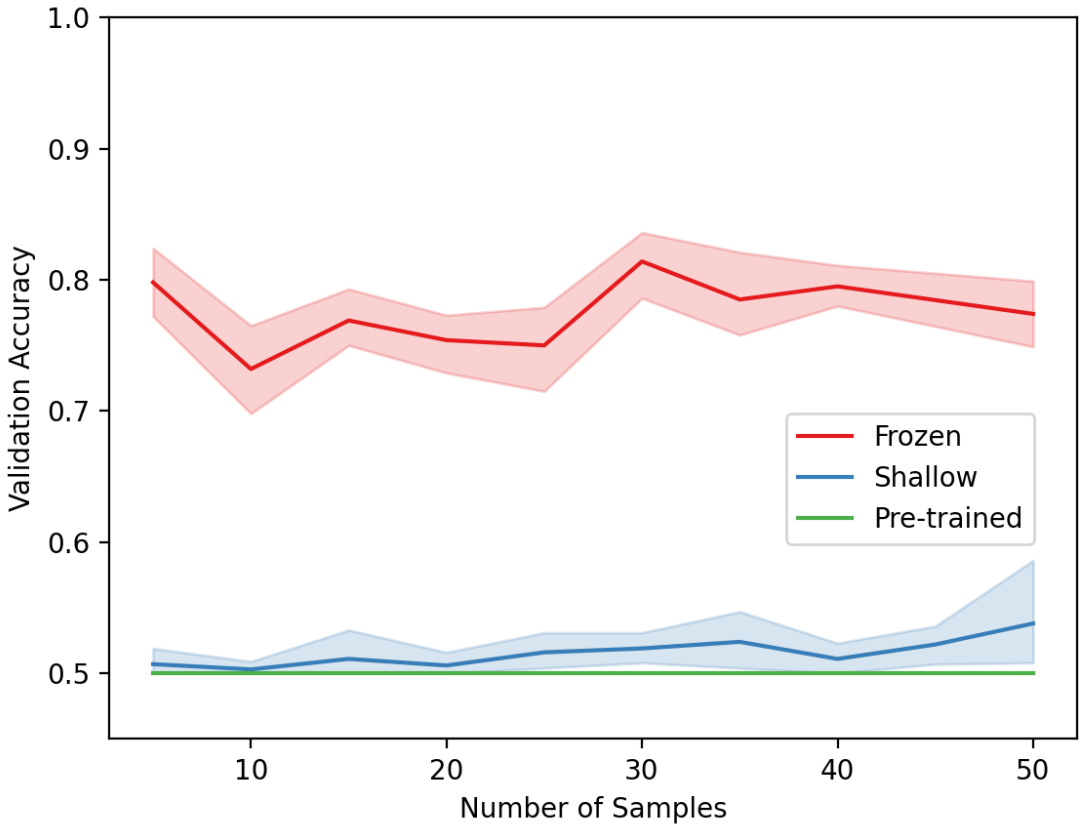


Figure 2.4: Accuracy achieved by each network at the corresponding sample sizes, from 5 to 50 samples in each UML category. For robustness, 5 trials were run for all training samples tested. The color bands indicate the distribution of results from the 5 trials.

We also compared the computational cost of training only the last layer of the frozen network to the entire unfrozen network. Training time for each model varies based on the number of epochs completed but generally, each one of these models can be fully trained in 30 minutes. The VGG model with frozen weights averages a little less than half a second faster, per epoch, than the

VGG model training all layers. The difference results from less computations required during the backpropagation of errors in models with frozen weights. As the dataset increases in size one can expect the difference in time between the two models to increase as more batches are completed per epoch. We can also compare our computation time to the computation time needed to train the original VGG-16. No doubt the difference in dataset size has an effect in reducing computation time, as the original network was trained on the large ImageNet dataset, but so would the number of trainable parameters. Simonyan and Zisserman, the creators of the VGG network, report that training a single network took 2-3 weeks depending on the specific architecture [15].

2.4.1 Class Activation Mapping

Using the Keras Visualization Toolkit [6], we produce CAM results for one UML sequence diagram and one class diagram. CAM results are shown in both Figure 2.5 and Figure 2.6 for the frozen VGG-16 network trained on 1800 sample images from each class. CAM produces a heat map highlighting the regions most heavily weighted by the network. We are able to see clearly that the network learns features specific to sequence and class diagrams. Specifically, in class diagrams, the boxes containing class attributes and methods have been highlighted. Conversely, in sequence diagrams, the vertical lifelines are more significant.

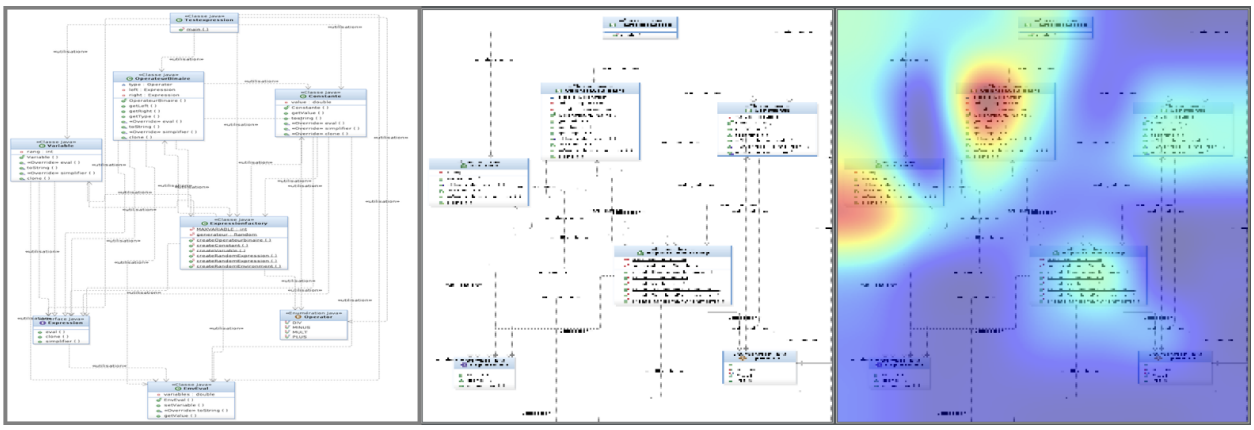


Figure 2.5: CAM result for a selected UML class diagram, original image on the left, resized image in the middle, and heatmap indicating significant features on the right

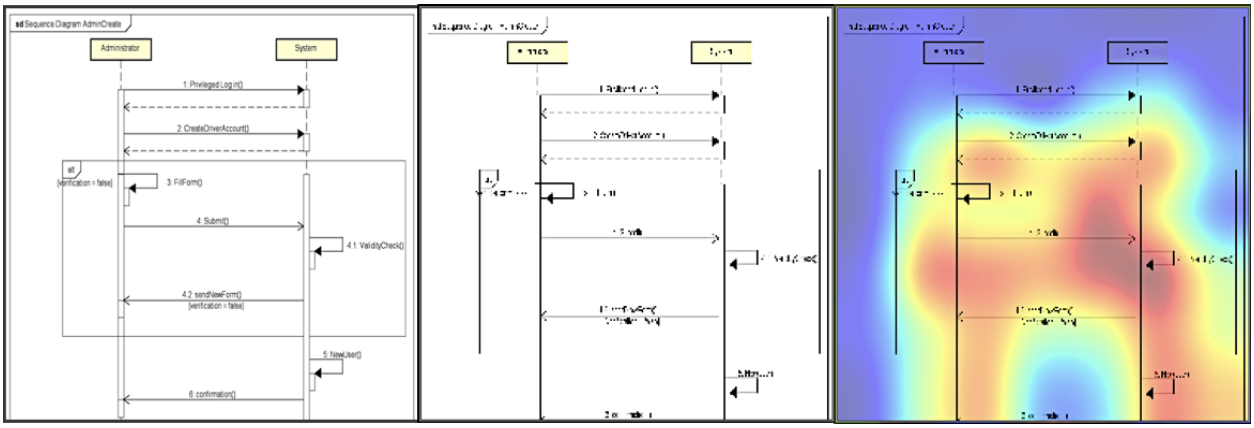


Figure 2.6: CAM result for a selected UML sequence diagram, original image on the left, resized image in the middle, and heatmap indicating significant features on the right

2.5 Related Works

The classification of UML diagrams has been studied through a variety of machine learning techniques. Ho-Quang et al. [25] proposed a logistic regression model using 19 of their 23 proposed features for classifying UML and non-UML class diagrams (CD). When trained on a corpus of 1300 images, their model achieved 96% accuracy for UML-CD and 91% of accuracy for non-UML CD. Years later, Ho-Quang et al. [25] furthered their work to differentiate between diagrams that were hand-made as part of the forward-looking development process (FwCD), and diagrams that were reverse engineered from the source code (RECD). However instead of classifying the images directly, the authors extract various features and implement a random forest model to achieve 90% accuracy in distinguishing the two types of class diagrams. In another study, using a corpus of 1300 UML and non-UML images, Hjaltason et al. [26] trained a support vector machine Support Vector Machine (SVM) with an average classification accuracy of 92.05% . Moreno et al. [27] conducted a similar study to classify web images as UML and non-UML class diagrams using a rule based approach. By extracting features from the images, in a corpus of 19000 web images, their algorithm reached an accuracy of 95%.

While we believe this is one of the first attempts to study the applicability of transfer learning to images within software engineering, transfer learning in general has been studied in many domains and aided in the development of powerful machine learning models. Authors in [28], propose the use of 'bellwethers', or the software project whose data yields the best predictions on all other projects. They argue that a simple transfer learner constructed from the bellwhether's data should be used as a baseline for future transfer learning work. In their study, they found that the simple transfer learner yielded comparable predictions to other more complex models. Effort estimation is just one area within the software domain where transfer learning has proven valuable. In an extension of previous work, Kocaguneli et al. [29], explore transfer learning in the field of effort estimation and for both the cross-company learning problem and cross time learning problem. Similarly, Ying et al. [30] also investigate transfer learning for cross-company defect prediction in software. Another study, takes one step further to include canonical correlation analysis into their study of cross-company defect prediction [31]. In physical applications, such as robotics, training samples can be especially costly, both in time and energy costs. In order to learn most efficiently while balancing these costs, transfer learning has been employed to predict the performance of physical systems under different configurations [32]. As a result, models do not need to be trained from scratch for each time and existing configurations can be adapted with few additional training examples.

Shin et al. [19] investigated the effectiveness of CNN architectures and transfer learning in detecting thoraco-abdominal lymph nodes and classifying interstitial lung disease from images. The authors achieve state-of-the art performance and find transfer learning to be beneficial despite the natural images used to train ImageNet being significantly different from medical images. Another study applied transfer learning to four medical imaging applications in 3 specialties including radiology, cardiology, and gastroenterology [33]. Their experiments transferred weights from ImageNet layer-wise, using none, a few, or many layers and found that transferring a few layers improved performance compared to training from scratch.

As stated previously, transfer learning in the space of software imagery was motivated by the work in [9]. Here the authors showed definitively that deep networks like VGG were unable to compete with smaller architectures when labeled data was sparse. A viable workaround was to create custom, shallower architectures that were compatible with available data volumes. The work presented here shows that off-the-shelf architectures can be used, but demand more efficient learning solutions - specifically the kinds produced via transfer learning.

The ultimate goal of the work in this chapter is to make deep learning and off-the-shelf convolutional architectures more available to empirical software engineering researchers who have a need to classify software artifacts. While large, labeled datasets are readily available for textual source code, for image-based artifacts such as UML, the curation of large volumes of training data continues to be a hurdle. This complicates the use of standard deep architectures such as VGG. The results achieved here indicate that transfer learning provides a path forward to researchers who wish to apply deep learning architectures to software artifact classification when only modest amounts of data are available. Specifically, pre-training with ImageNet using standard VGG architectures results in excellent classification performance of class and sequence diagrams despite the fact that the ImageNet dataset itself contains no examples of these artifacts. These benefits are in addition to those provided by transfer learning when massive training sets are available, in particular shorter model training times.

As with all work, there are some limitations to the experimental results presented here that are worth noting. First of all, experiments make use of only one data set based on UML. In future work, we will apply our transfer learning approach to other image-based software artifacts. Secondly, the classification task detailed here is binary, and discriminates only between class and sequence diagrams. It will be important to generalize this work to multi-class classification problems where only small amounts of training data are available. Finally, it would also be useful to assess the performance of datasets other than ImageNet as a basis for transfer learning.

3 A Fortran-Keras Deep Learning Bridge for Scientific Computing

3.1 Introduction

The Fortran programming language was originally developed in the 1950s and published in 1957. It was created to help programmers implement solutions for scientific and engineering problems on the IBM 704 computer, which at the time needed to be written in machine or assembly language. Fortran has been regarded as revolutionary and possibly one of the most influential software products in history [34]. Having evolved many times since its creation, with the most recent release in 2018, each version adds new features and capabilities. Fortran initially gained popularity and remains a widely used language due to its fast and efficient computational ability. Additionally, Fortran's strength is its backward compatibility, which allows modern compilers to build code written in the 60s and 70s.

Though not as popular as it once was, Fortran is still used in specialized fields, including oceanography, solid mechanics, computational physics, earthquake simulation, climate modeling, and aerospace. Because of Fortran's continued use, a great deal of legacy code and new code exists. Unfortunately, it is difficult to rewrite all existing code bases in more mainstream languages, due to their size and complexity. Therefore, when algorithms and extensive libraries are created in modern languages, backwards compatible methods must be developed to make them available in older legacy code, like Fortran.

In recent years, the rise of machine learning and deep learning has led to successful applications in various domains. Substantial improvements in the size of the training sets and available computing power have led to a new wave of implementations [35, 36]. In turn, this success has increased the usage and dissemination of deep learning. These methods have been

applied to a variety of domains, e.g., ranging from remote sensing [37, 38] to computer vision [39, 40, 21, 41, 42], and to games [43, 44]. Specifically, within scientific computing, many advancements have been achieved through the application of neural networks. Neural networks have been augmented with physically informed capabilities [45, 46], better suiting them for conservation restrictions. Learning partial differential equations [47, 48] has proved valuable in multiple scientific domains.

The success and popularity of deep learning has inspired the creation of powerful software libraries written in several modern programming languages. However, Fortran is not among the modern languages that benefit from these deep learning libraries. This absence leaves Fortran programmers with few options to implement deep neural networks.

The implementation of deep neural networks, in Fortran, may be achieved via two primary pathways. One solution is to rewrite all existing deep learning libraries in Fortran. The second solution is to leverage existing frameworks and bridge available functionalities to Fortran. The former is extremely arduous and time consuming, considering the size and scope of existing deep learning packages and the dizzying pace of their evolution [49, 50, 51]. The latter approach, which this chapter describes, is to allow users to leverage the power of existing frameworks while providing a bridge between paradigms where deep learning resources are plentiful and those where they are scarce. In this way, we can leverage aspects of currently available deep learning software libraries, like Keras [49], and bring them to large-scale scientific computing packages written in Fortran. To this end, we propose the Fortran-Keras Bridge (FKB) – A two-way bridge connecting models in Keras with ones available in Fortran. The source code is publicly available and can be found here: https://github.com/scientific-computing/FKB. We begin by reviewing existing Fortran projects that would benefit from the integration of FKB.

3.2 Fortran Projects

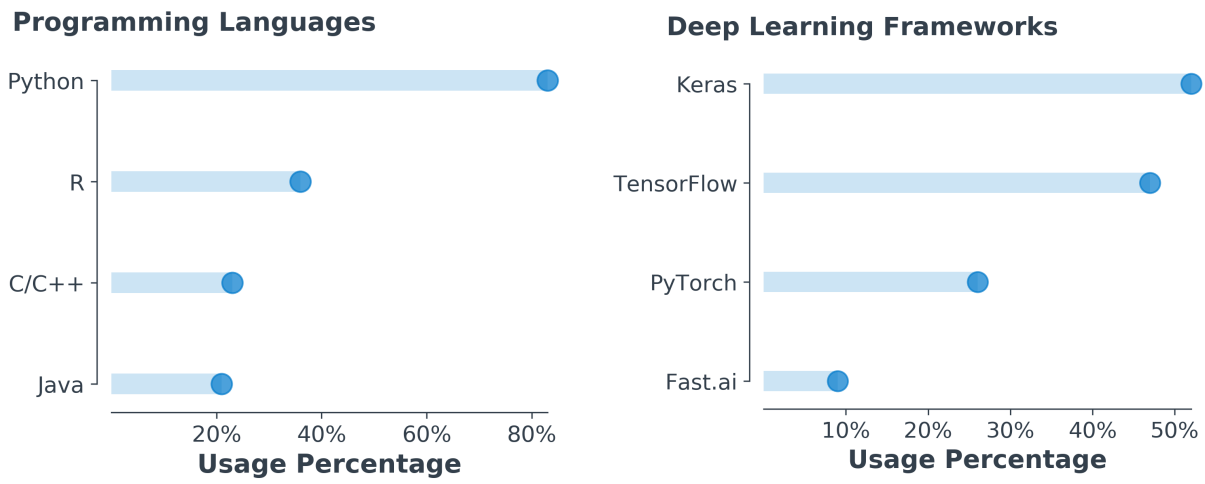
FKB can be integrated with many existing large-scale and computationally intensive projects written in Fortran. These projects will benefit from the easy integration of neural network models, which FKB makes possible.

For example, Fortran is used to do a great deal of work in climate and ocean modeling. For instance, the US-produced Community Earth System Model [52] is written in object-oriented Fortran-90; this is the most widely used climate model in the world. So are the other climate simulation codes used by the US Department of Energy [53] and the National Oceanographic and Atmospheric Administration's Geophysical Fluid Dynamics Laboratory [54]. Meanwhile, the Nucleus for European Modelling of the Ocean (NEMO) engine is used for studying ocean circulation problems on regional and global scales [55] and making future predictions, is also written in Fortran. The Hybrid Coordinate Ocean Model (HYCOM) [56], also used for ocean modeling, extends traditional ocean models to allow for a smooth transition from the deep ocean to coastal regimes. Researchers have also developed models for the modeling of waves and wind stress [57]. The Weather Research and Forecasting Model (WRF), is arguably the most widely used numerical weather prediction models for regional decision support [58]. Since its release in 2000, the number of WRF registrations has grown to over 36,000. WRF produces atmospheric simulations with support for special applications, including air chemistry, hydrology, wildland fires, hurricanes, and regional climate, and is again a Fortran-based model.

Fortran has found continued use in solid mechanics packages for implementing finite element methods. Popular packages such as ANSYS [59], ABAQUS [60], and LS-DYNA [61] are written in Fortran or accept Fortran subroutines. Similarly, in earthquake modeling, the SPECFEM3D [62] package leverages Fortran for simulations.

The list goes on. Code Saturne [63], developed by Électricité de France, and NEK5000 [64], are Fortran open-source computational fluid dynamics packages. Code_Saturne allows for user customization via Fortran subroutines, which is just one application domain for FKB. NEK5000 is actively used in the Center for Exascale Simulation of Advanced Reactors (CESAR) projects. Fortran has also been continually used for molecular modeling within chemistry and physics. The Chemistry at Harvard Macromolecular Mechanics (CHARMM) Development Project has produced a powerful molecular simulation program in Fortran [65]. This simulation program primarily targets biological systems but can also be used for inorganic materials. A similar tool, NWChem, has been developed by the Molecular Sciences Software Group at the Pacific Northwest National Laboratory [66]. NWChem is a computational chemistry software that includes quantum chemical and molecular dynamics functionalities. Within the molecular physics domain, Fluktuierende Kaskade (FLUKA) is a proprietary tool for calculations of particle transport and interactions with matter [67].

The models mentioned above and projects can leverage the FKB library to implement neural networks within their codebases. For example, neural networks have proven useful in modeling sea surface temperature cooling for typhoon forecasting [68]. Therefore the integration of FKB with tools like NEMO, HYCOM, or WRF models is a possibility. In a recent study of computational fluid dynamics, Ling et al. solve the Reynolds-averaged Navier-Stokes equations, similar to Code_Saturne and NEK5000. By implementing deep neural networks, the authors report that the architecture improved prediction accuracy [69]. Finally, the Fluka tool contains a wide range of molecular physics applications, including dosimetry calculations. Vega-Carrillo et al. have shown neural networks aided in the calculation of neutron doses [70]. For global climate simulation, there is proof that deep neural networks can offer skillful alternatives to assumption-prone approximations of sub-grid cloud and turbulence physics in the atmosphere [71, 72]. We hope that the FKB library enables Fortran users to expand their research and projects to include neural networks.



(a) (b)

Figure 3.1: (a) Usage of programming languages for machine learning and data science. Statistics are from the 2018 Kaggle ML & DS Survey [73]. (b) Usage metrics of deep learning frameworks. Statistics are from the 2019 Kaggle State of Data Science and Machine Learning report [74].

Having reviewed several Fortran based projects that can leverage FKB, we now introduce the two sides of this bridge. The following sections will develop the foundations on which to anchor each side of this two-way bridge. We start by introducing the deep learning anchor.

3.3 The Python Anchor (Deep Learning)

Many programming languages offer tools and libraries for implementing artificial neural networks. However, in recent years, Python has emerged as the clear favorite within this domain. Metrics in Figure 3.1a display Python's dominance. Python is used nearly 50% more than the second most popular language, R. Python's ubiquitous presence in machine learning makes it the obvious choice to leverage existing libraries for Fortran. The question then becomes, which available software library within Python, is best suited to bridge to Fortran?

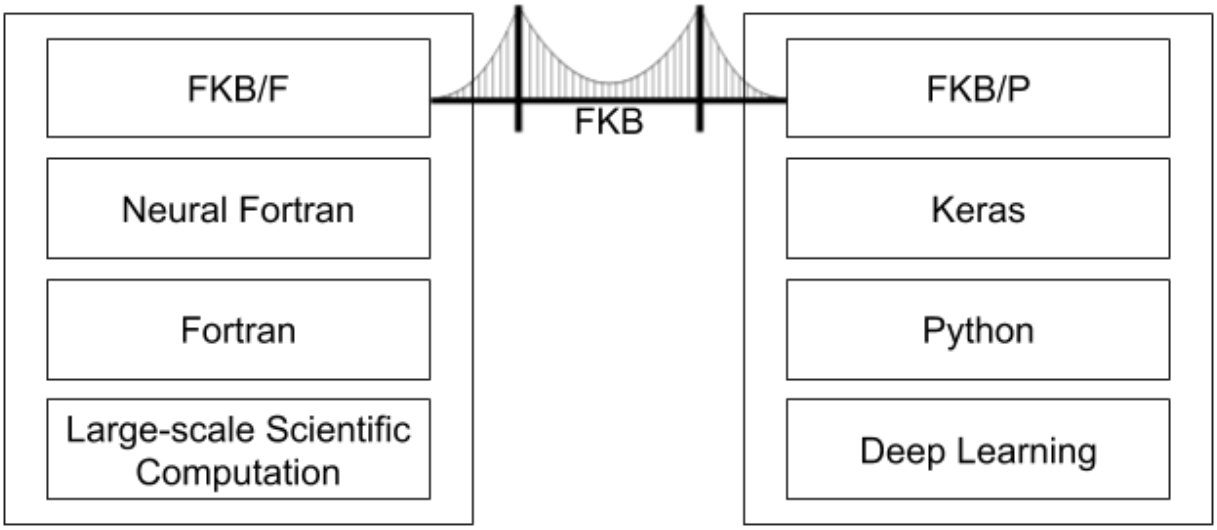


Figure 3.2: Positioning of FKB within Fortran and Python ecosystems.

Of the available deep learning libraries, Keras [49] is the most popular among practitioners (Figure 3.1b). Keras is an Application Programming Interface (API) built on top of Tensorflow [50], that provides users the ability to implement quickly, train, and test networks. This convenience encapsulates much of the low-level complexity one must manage when implementing deep networks from scratch. Keras abstracts many of the complicated aspects of Tensorflow while still providing customizability and ease of use. This combination makes Keras the first choice of many for deep learning applications. As a result of its popularity and ease of use, Keras is the clear choice on which to build one end of the two-way bridge.

Figure 3.2, depicts the positioning of the Python anchor, FKB/P, within the deep learning ecosystem. The Keras API leverages Python to build deep neural networks. FKB/P resides on top of Keras to access models produced from Keras and transmit them to the Fortran anchor, FKB/F. This structure allows for integration with Fortran applications that wish to leverage deep neural network architectures. Having described the deep learning anchor within Python, the next section develops the foundation for anchoring the bridge with Fortran.

3.4 The Fortran Anchor (Scientific Computing)

Several attempts have been made to implement neural networks in Fortran, with some success [75, 76, 77, 78, 79]. However, many implementations resort to hacking a single-use neural network by hand, or binding code from other languages [79]. Along these lines, one may consider accessing Python functionality directly from Fortran, by running a Python instance within Fortran. While providing flexibility and ease of use, this is vulnerable to extreme deficiencies in speed and computational resources as we found in preliminary tests. As a result, this solution becomes untenable for large-scale computation projects like the ones described in section 3.2. Another possible solution that could be pursued, is to call a machine learning model via a web API, as is common practice in industry production environments. This approach was not explored in this study. Although, we believe the same issue of speed would arise, especially in a setting like our case study, in which thousands of models are needed.

There are a small number of existing neural network libraries in Fortran [79, 80, 75]. The most recent and well developed library is Neural Fortran [75], a lightweight neural network library, written natively in Fortran. The Neural Fortran library provides the ability to implement artificial neural networks of arbitrary size with data-based parallelism. Additionally, in benchmark studies, Neural Fortran was shown to have comparable compute performance with Keras while maintaining a lower memory footprint. This library offers a foundation to anchor the Fortran side of the two-way bridge, FKB/F. By extending - and building on top of - Neural Fortran, we can convert Keras models to ones readily available in Fortran and implement them in existing Fortran projects.

The positioning of FKB within the scientific computing ecosystem is shown in Figure 3.2. The Fortran anchor, FKB/F, can use models originally constructed and trained in Keras, which can then be transferred to Fortran via FKB/P. To use these models, the Fortran side of FKB implements a neural network library. This portion of FKB can be used within large-scale scientific computation software, like the projects identified in section 3.2.

By leveraging FKB, it becomes seamless to train networks in Python and transfer them to Fortran, to run inside large scale simulations. Similarly, neural network models constructed in Fortran can be transferred to Python for additional analysis, expansion, and optimization - including hyperparameter searches using available tools in Python [81, 82, 83]. As both sides of the bridge have been properly introduced, the following section will describe the specific features and functionalities of FKB.

3.5 Features of FKB

Once a neural network is trained in high-level APIs like Keras, the practitioner has few practical avenues for using this model in Fortran-based projects. One approach may be to hard code network operations inside Fortran while manually moving parameters from the Keras model. Several examples of this can been seen in climate modeling [71, 72, 84, 85].

To provide one specific example, in [71], the authors trained a DNN to represent sub-grid cloud and convective energy transport processes, in Keras. To assess its credibility, they needed to test the DNN's two-way interactions when thousands of replicates of it were embedded within a coarse-resolution global atmospheric model, written in Fortran – neural network emulated clouds interacting with deterministic physical calculations of planetary geophysical fluid dynamics. As the global atmospheric simulator does not offer native neural network support, the authors hardcoded their DNN model into the global simulation software framework. This approach has obvious disadvantages. Every minor change made to the model in Keras requires rewriting the Fortran code. If one wishes to test a suite of models in Fortran, this approach becomes untenable.

As each network may require different hyperparameters and, as a result, necessitates rewriting and compiling the Fortran code for every new model. This process drastically limits the breadth of available models to be tested within the simulator. This bottleneck is currently a significant roadblock to ongoing debates in the climate simulation community, more broadly, about whether

or not to use DNN representations of subgrid physics in next-generation climate modeling. Insufficient testing of diverse candidate Neural Networks (NNs) means that little is known about how minor imperfections in the fit of one NN can amplify when the NN is coupled to fluid dynamics, which is just beginning to be explored [86].

These issues demand a solution, in the form of a bridge between Keras and Fortran. The FKB software solves these issues via two key elements. First, it provides a neural network library implemented in Fortran (FKB/F). Second, it offers the ability to parse existing Keras models into formats consistent with the Fortran neural network library (FKB/P). As a result, users can switch, seamlessly, back and forth between Python and Fortran. This context provides a way for iterative neural network tuning (Python) and testing (Fortran), with a simple way to translate between the two software environments. Additionally, FKB offers currently unavailable Fortran specific features for neural networks. It will be useful to highlight those new features while documenting the format to which FKB adheres. The following subsections describe the Python and Fortran anchors' features, FKB/P and FKB/F, respectively.

3.5.1 FKB/P

Keras models - once built, trained, and saved - are stored in Hierarchical Data Format version 5 (HDF5) files. These files contain the network architecture, weights, biases, and additional information - optimizers, learning rates, gradients, etc. From the HDF5 file, FKB/P parses the network architecture, extracting the number of layers, activation functions, nodes per layer, and all weights and biases. This information is converted to match the Fortran neural network configuration in FKB/F. This allows users to build an equivalent network in Fortran, which can easily be loaded and used within a Fortran environment. If any modifications to the model are made inside Fortran, FKB/P will parse this back into the equivalent HDF5 file to be used in Keras once again.

On the other hand, networks may be initially constructed in Fortran. After initial training and testing, a user can switch to Keras for further evaluation. From Keras, users can conduct additional testing or hyperparameter tuning where these tools are readily available [81].

The ability to seamlessly pass neural network architectures between Python and Fortran is essential for any practitioner working in this space. This bridge allows users to take advantage of the high-level Keras API - training on computationally efficient Graphics Processing Units (GPUs) - then to insert their trained model into a Fortran codebase. The functionality provided bridges the chasm between Keras and Fortran.

3.5.2 FKB/F

The Fortran anchor of FKB leverages and extends the original Neural Fortran library. Below we introduce newly implemented features to make Neural Fortran more flexible and able to communicate on the two-way bridge.

Custom Layers

To implement neural networks in Fortran, FKB leverages and extends the Neural Fortran library [75]. The prototype Neural Fortran library format that we build on was only capable of implementing a fully connected layer. Forward and backward operations occurred outside this layer - in the network module. An example of this is shown in Listing 3.1. From the listing, one can observe hard-coded matrix multiplication of layer weights, the addition of biases, and the activation functions inside the network module. This network-level subroutine accesses and modifies individual layer attributes. This rigid format is inconsistent with modern neural network implementation paradigms [49, 50, 51], but it makes it impossible to implement other layers or custom operations. To increase the library's flexibility, operations must be encapsulated inside the layer, consistent with current practice.

```
pure subroutine fwdprop(self, x)
    ! Performs the forward propagation and stores arguments to activation
    ! functions and activations themselves for use in backprop.
    class(network_type), intent(in out) :: self
    real(rk), intent(in) :: x(:)
    integer(ik) :: n
    associate(layers => self % layers)
        layers(1) % a = x
        do n = 2, size(layers)
            layers(n) % z = matmul(transpose(layers(n-1) % w), layers(n-1) % a) &
                + layers(n) % b
            layers(n) % a = self % layers(n) % activation(layers(n) % z)
        end do
    end associate
end subroutine fwdprop
```

Listing 3.1: Original code from [75]. Layer operations occur inside the network module, limiting flexibility.

In FKB we introduce an extendable layer type module (Listing 3.2). To implement a layer, one simply extends the layer type and specifies the construction of the forward and backward functions. Adhering to this format offers several advantages. By restructuring the format of the library, we offer the ability to implement arbitrary layers. Additionally, in the network module, all layers are stored in an array of pointers. This leads to the encapsulated version shown in Listing 3.2 wherein a forward pass, in the network module, calls the layer-specific forward function. In this way, all operations are confined to the layer module, and the output from one layer is passed as input to the next.

```
function output(self, input) result(last_layer_output)
    ...
    ! iterate through layers passing activation forward
    do n = 1, size(layers)
        call layers(n) % p % forward(layers(n-1) % p % o)
    end do
    ! get output from last layer
    last_layer_output = layers(size(layers)) % p % o
end function output
```

Listing 3.2: Forward pass in the FKB network module. Each layer simply calls its own forward function. The technical operations occur within each layer.

FKB supports fully connected or dense layers, dropout [87, 88], and batch normalization [89]. Shown in Listing 3.3 is an example of extending the layer_type to implement a Batch Normalization layer. This format translates to increased functionality and customizability to the user. As a result, more standard layers from Keras are available, while giving users the flexibility to implement their own custom operations.

```
! BatchNorm layer - extends from base layer_type
!   Implements batch normalization
type, extends(layer_type) :: BatchNorm
    ! epsilon parameter
    real(rk) :: epsilon
contains
    procedure, public, pass(self) :: forward => batchnorm_forward
```

```
    procedure, public, pass(self) :: backward => batchnorm_backward
end type BatchNorm
```

Listing 3.3: Example of extending the layer_type to implement Batch Normalization

Training in Fortran

It is necessary to distinguish between the terms *offline* versus *online* for the following section. These terms serve to distinguish two different settings in which a neural network can be used in a Fortran computing package. Both settings can make use of historical or simulated data to train an artificial network. The distinguishing feature is how the predictions of a model are used. In an online setting, predictions from the model are used to evolve a physical process. The predictions at one time step effect how the system acts at the following time step. As a result, inputs to the model will change based on how the model acted in the past. In offline settings, this is not the case. Predictions made in the past do not affect the input to the model in the future.

In many cases, offline training may be sufficient to learn a model, if enough prior data is available. However, in some cases, online training may be the method of choice. To this end, FKB is equipped to handle backpropagation for gradient descent optimization of a specified cost function.

The layer encapsulation mentioned above of forward and backward operations (Section 3.5.2) becomes extremely valuable in training. Instead of all computations occurring within the network module [75], they are contained in layer-specific functions. Much like the forward pass, backward operations occur in the layer. In this fashion, each layer is responsible for computing its gradients with respect to its parameters and returning the gradient with respect to the layer below it.

Online training can serve a variety of purposes. First, a neural network model may be learned entirely in Fortran, based on the evolving state variables during the integration of a physical dynamical system simulation, and then transferred to Keras after the fact. In this setting, the ground truth, from the simulator, is passed to the network for it to calculate its errors and update its parameters accordingly through backpropagation. Second, online training could serve to provide gentle corrections to an imperfect pretrained model, for instance, to hedge against the amplification of its imperfections that are only revealed once the NN is coupled to other physical calculations. Here a model is trained offline in Keras and transferred to Fortran (Section 3.5.1). In some cases, for a variety of reasons, the offline training data may have a differing distribution than that of the online data. In such a setting, it proves beneficial to offer slight corrections to the network. Finally, a secondary model may be constructed to learn and compensate for the deficiencies in the primary model. In this way, the two networks work together to balance out any instability issues.

The ease of use and proper format directly results from the encapsulation of layer operations. Online training offers a solution to tackle a suite of potential problems. As a result, models may be updated with slight corrections or learned entirely online.

Custom Loss Functions

In many applications, practitioners may wish to optimize a unique quantity - a function other than a mean squared error or cross-entropy. This is common when target variables interact or additional information is known about their relationship in a desired application. For example, in modeling any physical system, predictions from a neural network must not violate physical constraints - energy cannot be created or destroyed in the system. To satisfy this restriction, a loss function can be written to quantify the amount of violation of physical properties. This construction can then be minimized to alleviate constraint infractions [46].

The implementation of custom loss functions is standard for high-level APIs like Keras, Tensorflow, and PyTorch to provide this ability in their codebase [49, 50, 51]. As FKB is designed for those working in the physical sciences where environmental, physical, or application-specific constraints are common, it provides the ability to implement custom loss functions. To take advantage of this functionality, users must implement their desired loss function, just as they would in Keras. As FKB does not provide automatic differentiation, the derivatives with respect to the input are also required for training. Once these functions have been specified they can be dropped into the existing framework and run normally, much like Keras.

```
real(rk) function crossentropy_loss(self, y_true, y_pred)
    ! Given predicted and expected output, returns the scalar loss
    class(network_type), intent(in out) :: self
    real(rk), intent(in) :: y_true(:), y_pred(:)
    loss = - sum(y_true * log(y_pred))
end function loss

function d_crossentropy_loss(self, y_true, y_pred) result(loss)
    ! Given predicted and expected output
    ! returns the loss with respect to softmax input
    class(network_type), intent(in out) :: self
    real(rk), intent(in) :: y_true(:), y_pred(:)
    real(rk), allocatable :: loss(:)
    loss = y_pred - y_true
end function d_loss
```

Listing 3.4: Implementation of crossentropy loss function and the corresponding derivation with respect to the input logits.

This capability is demonstrated through the implementation of the cross-entropy loss function in Listing 3.4. To implement this previously unavailable loss function, we first declare two functions. First, the cross-entropy scalar loss is defined. Second, the loss with respect to the input logits is derived. These two functions are then referenced as the loss and d_loss, respectively. By providing this functionality, users may leverage a variety of loss functions that can be used to minimize application-specific quantities. Once described, they may be included with the existing framework and used during online training.

Ensembles

Ensembles consist of different models, each trained on the same, or bootstrapped, data. The output of the ensemble will be an average of all its member's predictions. In machine learning, ensembles of models typically perform better than any one of its members alone. The ensemble strategy exploits the fact that each model will make different errors. Therefore, when averaged together, these predictions become more accurate, as certain errors get smoothed out. A consensus from machine learning practitioners is ensembling gives 1-2% improvement in performance [90].

As a result of this averaging, ensembles provide a boost in performance as well as additional robustness. In domains where physical constraint violations yield stability issues, ensembles may be applied to dampen these problems. By averaging across many networks, the instability of any one model will be drastically reduced in the presence of more sound predictions.

The functionality provided requires the user to specify a directory that contains the models of interest and a desired amount of noise. The ensemble type will read in each model and construct a network corresponding to each of them. To get a prediction from the ensemble, an input vector is passed to it. For non-zero amounts of noise, Gaussian noise is applied to the input vector each time it is passed to an ensemble member. This allows each member to see a slightly different variant of the input, increasing the robustness of prediction around that point. This operation runs

in parallel using OpenMP, where each network can be given its thread to expedite computation; such an approach could easily be adapted via OpenACC for GPU-based threading of large ensemble network calculations. Following the computation, the predictions are averaged together, and the final output is given.

3.6 Case Study

The following section provides a case study demonstrating an application of FKB to experimental next-generation climate modeling. The Superparameterized Community Atmospheric Model version 3.0 (SPCAM3) is used for all simulations in this study. SuperParameterization is an approach that confronts the decades-long problem of representing subgrid cloud physics in climate models by embedding thousands of limited-domain explicit sub-models of moist convection within a conventional planetary-scale model of the large scale atmosphere [91, 92, 93, 94]. This approach tends to involve two orders of magnitude more computational intensity per unit area of the simulated earth, but recently Rasp et al. used a deep neural network to emulate all of the expensive subgrid Cloud Resolving Model (CRM) and their influence on the planetary host at drastically reduced computational expense [71]. This study, along with others in the emerging climate modeling literature [72] have demonstrated the potential advantages of a data-driven approach for addressing the critical unresolved effects of clouds and convection on planetary climate, as compared to previous, heuristic based, approximations to subgrid physics. However, the idea of emulating turbulence in climate simulation is still an emerging one, with unclear trade-offs, including frequent instabilities when NN emulators are coupled with fluid dynamics, which the community is seeking to learn how to control [72]. It has even been questioned whether the offline skill of such emulators, during their training, is predictive of their online performance [95, 96], an important open question.

These questions are understudied primarily due to the lack of the simple software interface that FKB now enables for climate scientists to test diverse candidate neural networks, and ensembles within planetary climate models. To illustrate an advance on this front we now apply FKB to shed new light on two related questions currently in debate:

1. Does offline performance translate to online model performance [95, 96]?
2. Which neural network hyperparameters most affect online performance?

Using FKB, the study can be broken into two stages. First, a suite of 108 candidate neural network models of convection are trained, via Keras, on simulated data from the SPCAM3. Second, the models are converted to Fortran and run online (i.e. coupled to planetary fluid dynamics) in the SPCAM3 simulator. The number of steps serves as a preliminary metric of performance until catastrophic failure. It is clear that in the absence of the FKB library, running hundreds of candidate neural network submodels of convection within the Fortran based model of the rest of the planet's atmosphere would be nearly impossible. This is due to the fact that each network contains various hyperparameters, each with different weights and biases learned during training, including layer-specific properties such as optional use of dropout or batch-normalization. In order to leverage the FKB library with SPCAM3, we simply compile the neural network library in advance and link it to the compilation of SPCAM3. Documentation and steps for implementation of this case study are provided here:

https://github.com/scientific-computing/FKB/blob/master/SPCAM_Instructions.md.

The input to this neural network model is a 94-dimensional vector. Features include vertically resolved vectors representing the large scale (host model) temperature, humidity, meridional wind vertical structure, surface pressure, incoming solar radiation, sensible heat flux, and latent heat flux scalars. The output of the network is a 65-dimensional vector composed of the embedded models' influence on their host - i.e. the sum of the CRM and radiative heating rates, the CRM moistening rate, the net radiative fluxes at the top of the atmosphere and surface of the earth, and the precipitation.

Name	Options	Parameter Type
Batch Normalization	[yes, no]	Choice
Dropout	[0 - 0.25]	Continuous
Leaky ReLU coefficient	[0 - 0.4]	Continuous
Learning Rate	[0.00001 - 0.01]	Continuous (log)
Nodes per Layer	[128,256,512]	Discrete
Number of Layers	[4 - 11]	Discrete
Optimizer	[Adam, RMSProp, SGD]	Choice

Table 3.1: SHERPA Hyperparameter Space

The training data used here are challenging to fit, as they come from an enhanced version of the CRM training data that was originally studied by [71]. In superparameterized simulations, one can control the degrees of freedom of the interior resolved scale through the room available for interesting forms of sub-grid storm organization to form. One can control the physical extent (i.e. number of columns used in) each embedded CRM array [97]. In [71], CRM arrays with only 8 columns (32-km extent, given the 4-km horizontal resolution) were used. Here we quadruple the extent (from 32 km to 128 km, i.e. from 8-columns to 32-columns) to improve its physical realism. Despite several attempts, these data have never been fit successfully. NNs trained from the enriched data tend to produce crashes within just a few simulated weeks after they are embedded in the climate model (see discussion of "NN-unstable" by [86] for details).

Our working hypothesis is that historical failures in free-running tests when emulators are trained on higher quality CRM training data reflect a broader issue of insufficient hyperparameter tuning in climate model applications. To address this, we conducted neural network optimization via a random search using SHERPA [81], a Python library for hyperparameter tuning. We detail the hyperparameters of interest in Table 3.1, as well as the range of available options during the search. The hyperparameters of interest consisted of whether or not to use batch normalization,

the amount of dropout, the leaky ReLU coefficient, learning rate, nodes per layer, the number of layers, and the optimizer. The random search algorithm has the advantage of making no assumptions about the structure of the hyperparameter search problem and is ideal for exploring a variety of settings.

We attained 108 candidate neural network model configurations, each trained for 25 epochs with early stopping monitoring the validation loss. Following the offline training stage, the neural network models were converted into their Fortran counterparts and ran inside SPCAM3. We underscore that this critical step would have been prohibitive using standard tools that have required manual translation of each candidate model. However, by leveraging the FKB library, each model was loaded independently into Fortran and run as the subgrid physics emulator inside SPCAM3's host planetary model, of the large-scale atmospheric state. Each model was coupled to fluid dynamics, to run a wide ensemble of prognostic tests across an unprecedented diversity of candidate neural network architectures. Each of the one hundred and eight candidate neural network models - with their various numbers of layers, layer-specific settings (batch-normalization, relu magnitude, etc), nodes per layer, weights, and biases - were run online, all without rewriting any Fortran code.

In order to address the first question and evaluate a neural network model's performance, we compare its validation MSE during training with the time-to-failure of the online tests in which 8,192 instances of the NN, spaced at regular intervals around the globe, are coupled interactively to their host global atmospheric model of large scale geophysical fluid dynamics. This yields Figure 3.4a, which sheds new light on the offline vs. online relationship.

The results in this figure demonstrate a relationship between offline validation error and online performance. There is a distinct, negative, relationship between offline MSE and online stability (Spearman correlation of -0.73; $p = 4.961e^{-19}$. Intriguingly, the mean-squared error loss of our multi-layer perceptron is a reasonable predictor of stability once coupled to the climate model,

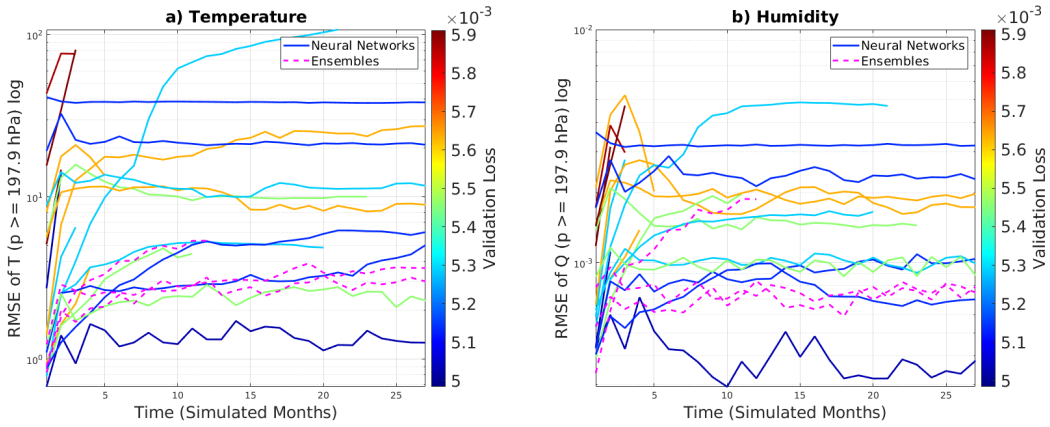


Figure 3.3: The time-evolution of the tropospheric (a) temperature and (b) humidity biases, colorized by the offline validation error

insofar as the time-to-failure is concerned. This finding is interesting in the context of the recent speculation by [95] that such a relationship might not exist using similar NNs in a similar setting, as well as the comments by [96] about similar incongruities even in reduced-order dynamical systems when emulated with generative adversarial networks.

Of course, stability alone is a necessary but not a sufficient condition of prognostic success, which also requires an in-depth analysis of biases in the simulated climate. Figure 3.3 shows the time-evolution of the tropospheric temperature and humidity biases, colorized by the offline validation error. These metrics reveal that although our search has uncovered many runs that are "stable" - can run without catastrophically crashing for several months - most of these runs would not be very useful in an operational setting. Almost all NNs exhibit major errors in the simulated climate, having drifted to erroneous attractors with root-mean-square errors in temperature frequently above 10 K. However, the NN that produced the best offline validation error stands out as having the combined desired qualities of stability and skill with temperature biases of less than 2 K, competitive with [71]. Interestingly, coupling instead to the ensemble mean of a few of the best-ranked models (magenta dashed lines) does not outperform coupling to the best fit model, the value of having found it using SHERPA (Figure 3.3).

In short, we have produced a successful coupled simulation that was particularly challenging without formal hyper-parameter tuning and FKB. This result suggests that sufficient hyperparameter tuning may be critical to solving chronic instability in climate model applications of DNNs for subgrid physics.

The second question naturally arises as to which of the hyperparameters are most impactful to the online performance. To assess this, Figure 3.4b-i decomposes the sensitivity of the baseline relationship to individual hyperparameter choices. The choice of optimizer is shown to correlate most strongly with online performance (Figure 3.4i). This finding is confirmed by Spearman values, shown in Table 3.2. The optimizer hyperparameter has the largest absolute correlation value with online performance. No other hyperparameter shows as clear a distinction in correlation that is evident in the choice of optimizer, including the network depth and total number of parameters, which are known to be important to offline fits for this problem [98], but are surprisingly not as predictive of coupled skill as the choice of optimizer, whose impact has not previously been isolated (for this application).

Further investigation into the specific optimizer used, reveals the Stochastic Gradient Descent (SGD) optimizer to perform poorly; NNs fit with SGD never run longer than 1,000 steps when coupled online (Figure 3.4i). Again the visual intuition from Figure 3.4i is confirmed by Spearman correlation values. SGD, Adam, and Root Mean Square Propogation (RMSProp) have Spearman values of -0.6670, 0.5936, 0.0586 respectively. These values demonstrate that the use of SGD is negatively correlated with online performance, whereas Adam positively correlates with online performance. This result leads one to speculate that increased improvements in online skill may be realized from more advanced optimizers with enhanced gradient update schedules.

Finally, after answering the two questions motivating this case study, we can compare the results of the best performing model with that of previously published models of [71] when applied to the challenging limit of CRMs with 32-km horizontal extent. The model proposed by Rasp et al. was a single deep neural network. The hyperparameter space of this model was not fully explored online in large part due to the laborious process required to transfer those models

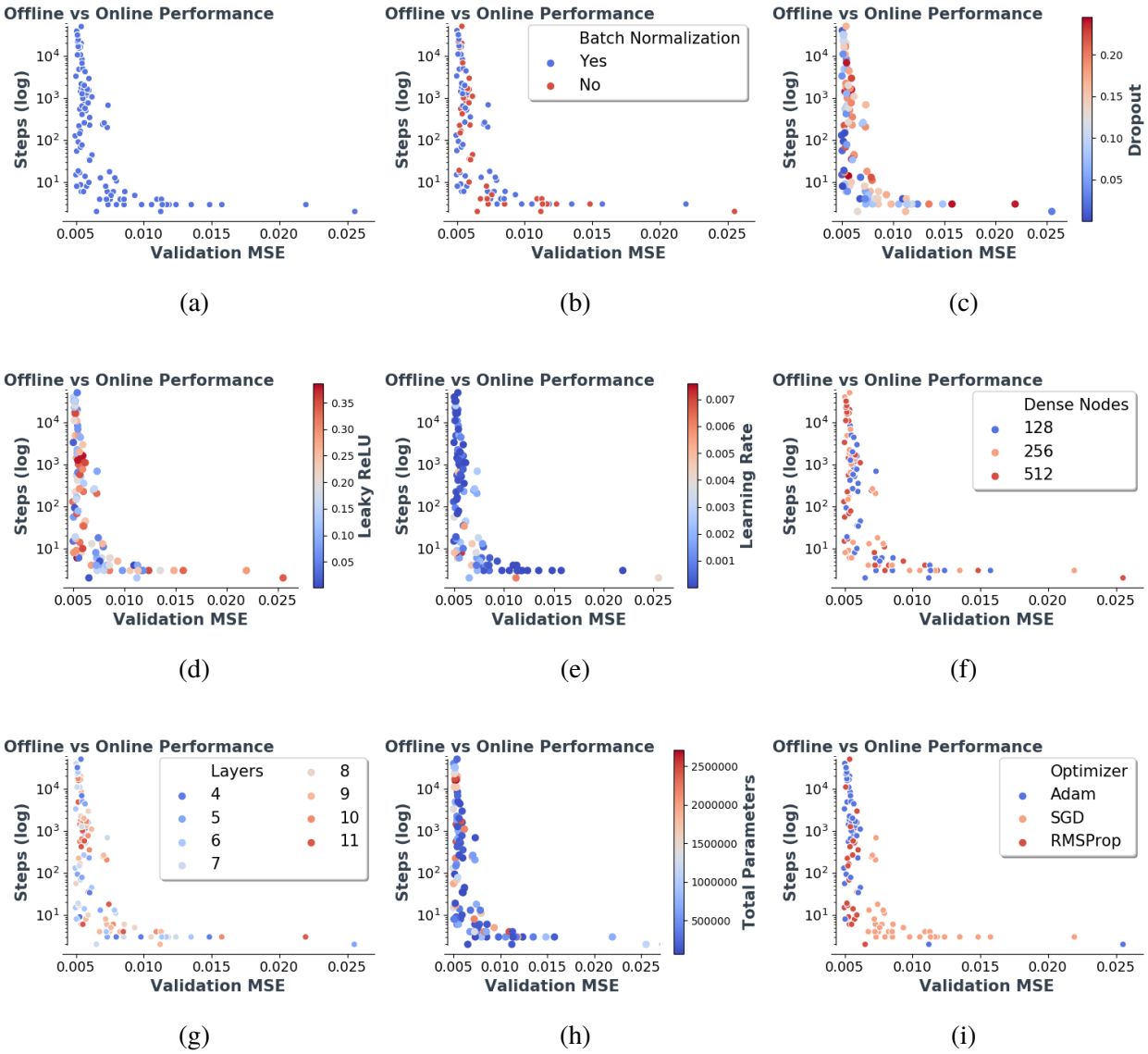


Figure 3.4: Offline performance - validation mean squared error (MSE) - vs online performance - number of steps until crash. (a) All models. (b) By batch normalization usage. (c) By Dropout amount. (d) By leaky ReLU coefficient. (e) By learning rate. (f) By number of dense nodes per layer. (g) By number of layers. (h) By total number of model parameters. (i) By optimizer type.

	Correlation	P-Value
BatchNorm	0.0859	3.7896e-01
Dropout	0.1919	4.7591e-02
Leaky ReLU	0.0055	9.5465e-01
Learning Rate	-0.2087	3.0923e-02
Dense Nodes	0.1427	1.4249e-01
Layers	0.0410	6.7491e-01
Optimizer	**-0.6998**	5.0177e-17
Parameters	0.1528	1.1609e-01

Table 3.2: Spearman correlation of corresponding hyperparameter with online performance, and associated p-value.

into Fortran. The Rasp et al. model (provided by the authors) ran for 128 steps before crashing due to instability issues. The five best models achieved in this study ran to completion of a 5-year simulation, i.e. for 87,840 steps; of these, two of the five models further exhibited root-mean-square errors in simulated tropospheric temperature of less than 2 degrees Celsius. This dramatic improvement in stability is a direct result of the ease with which a wide variety of models (identified by SHERPA) can be transferred between Python and Fortran (thanks to FKB). We also note that this method is preferable to another approach that was recently proposed to begin stabilizing the same model, through small-amplitude Gaussian input perturbation [86] - a strategy that, while promising, adds computational expense and introduces out-of-sample extrapolation issues that can be avoided with the brute-force optimization and wide-ensemble prognostic testing path to stabilization we have outlined here.

This case study has investigated two closely entangled questions: 1) Does offline performance correspond to online model performance? 2) What neural network hyperparameters most effect online performance? Both of these questions have been answered by leveraging the FKB library. The library offers the ability to expeditiously transfer models trained in Keras to Fortran, where they may be run online in existing simulators. In the absence of FKB, neither one of these questions could be approached without unreasonable human intervention, as the operational target is a climate model with over a hundred thousand lines of code written in Fortran.

4 An Information-Theoretic Analysis of Scientific Computing Software with Unsupervised Machine Learning

4.1 Introduction

The proprietary language, Matrix Laboratory (MATLAB) [99], has been a language of choice for those across industry and academia with backgrounds in engineering, science, and economics. MATLAB is a multi-paradigm numerical computing environment that offers a plethora of graphics functions, dedicated statistics functions, and a core focus on matrix operations. An open source competitor of MATLAB is the R programming language [100], which, despite it's appearance in 1993, has recently gained favor due to its breadth of statistical functionality, as well as its ability to easily integrate with more performance-minded (compiled) languages, such as C++ and Fortran. Moreover, many extensions are available to integrate with high performance techniques, including GPU-based computing, which paves the way for state-of-the-art machine learning algorithms (e.g. deep learning). Additionally, R has become a favored programming language for academic research, including some statistics journals, which require that R code used for analysis be made available along with data to maximize reproducibility of results. Given these considerations, it is no surprise that a recent survey conducted by the Institute of Electrical and Electronics Engineers (IEEE) reported R as the fifth most popular programming language, substantially above other scientific computing languages such as MATLAB and Scala, which occupy places fourteen and fifteen, respectively [101].

Despite its increasing popularity and the availability of large repositories of open source code, R, and statistical computing languages in general, have received relatively little attention from the empirical software engineering community. While this is ironic in the sense that many of the tools used to mine software repositories are implemented in these languages (including the tools and algorithms used in this paper), this also creates opportunities for new research questions. One such question is what are the typical functions of scientific computing languages, and how are they different from those found in other domains, such as enterprise computing applications? Further, how does the application of these functions across source code packages compare between open-source and enterprise computing? Though basic, answering these questions can help shed light on the nuances of scientific computing software and where it fits in the landscape of open source software repositories supporting computational science.

In this chapter we apply topic modeling, in particular LDA [11], to identify functional concepts from over 10,000 R packages and 27,000 MATLAB modules in order to compare how these concepts are applied across their respective languages. The results of our analysis provide insight into the underlying characteristics of scientific computing algorithms in general, with potential to guide further research in this area.

4.2 Data

The Comprehensive R Archive Network (CRAN) presently consists of 16,054 unique R packages [102]. This study examines 10,051 of these packages. The requirements for submitting R packages are documented on CRAN, as well as the extensive review process each package undergoes before it is added to the repository. Each package is composed of R files, with varying amounts of C, and C++ files for performance optimizations. In addition to R and MATLAB, we include these C++ files in our topic analysis.

Despite MATLAB's proprietary nature, over 38,000 open source packages are publicly available via the MathWorks File Exchange [103]. Our collection of MATLAB code consists of 27,130 modules, written entirely in the MATLAB programming language. Both the MathWorks File Exchange and CRAN represent a diverse collection of functionality geared toward scientific and high performance computing.

4.3 Methods

LDA-based approaches to modeling software repositories have been shown to be more effective than non-statistical techniques [104]. Since its first application by Linstead and colleagues [105], LDA and similar topic models have been used for a wide variety of software analysis applications [106, 107]. This includes the work in [108], which proposes that the topics identified by LDA correspond to aspects, and that information-theoretic techniques such as entropy analysis can be used to measure topic distributions across software repositories.

To prepare our corpus for LDA, we parsed the source code files, filtering stop-words. We applied standard naming heuristics to split camel case and underscore identifiers within the corpus. Although stemming is commonly leveraged in topic modeling research [109] to prevent words with common roots from diluting the topics produced (i.e. "model" and "models" being considered two separate tokens), in the context of software packages a single letter could significantly change the function of that package and thus we chose to omit it from our processing. Table 4.1 shows the resulting vocabularies for each respective language, with their corresponding number of tokens. This input was processed using the LDA implementation provided by the R package, Mallet [110], a popular toolbox for topic modeling. The number of topics to be extracted was determined empirically by creating topic models of various sizes, and assessing the results for human interpretability. The Dirichlet priors on the document-topic and topic-word distributions were determined via Mallet's built-in hyperparameter optimization routine rather than using the default heuristics for symmetric priors. The importance of this has recently been reported in the empirical software engineering literature [111].

Language	File Count	Lines of Code	Tokens
MATLAB	187,938	11,760,856	31,432,343
R	178,320	14,949,905	56,948,438

Table 4.1: This table shows the number of files, total lines of code, and number of unique tokens for topic modeling for MATLAB and R.

Once a model is trained using LDA, the degree to which topics are spread across documents (source files) and to which words (tokens) are spread across topics can be measured qualitatively via information-theoretic techniques as first proposed in [108]. Specifically, entropy scores were calculated for each topic. These scores represent the probability of documents given a topic by normalizing the frequency of a topic across all documents in the model and then calculating the entropy of that distribution. For these document entropies, a low entropy indicates that the topic is concentrated in a few documents while a high entropy indicates a more even spread across many documents. We measure topic uniformity using Mallet's built in method, which is based on calculating the Kullback-Leibler divergence to measure the distance from a topic's distribution over words to a uniform distribution.

4.4 Results

4.4.1 Topic Modeling: R Files

Table 4.2 provides a sample of the LDA topics produced by our model when trained on exclusively R packages collected from CRAN. The topics produced include applications both non-specific and specific to R. Topic 26, for example, illustrates the use of R to produce summary statistics on existing data sets (”par”, ”lower”, ”upper”, ”dist”, ”distribution”) while, topics 14 and 34 demonstrate R functionality associated with api usage (”url”, ”query”, ”api”, ”json”, ”status”) and I/O operations (”file”, ”read”, ”write”, ”header”, ”append”). Extending beyond these fundamental

concepts are topics that demonstrate the specific domains in which R is commonly applied. Topic 28 alludes to DNA sequencing (”gene”, ”data”, ”verbose”, ”ontology”, ”flag”). Topic 32 demonstrates processing light spectrum (”spec”, ”wave”, ”frequency”, ”seq”, ”matrix”) and topic 59 shows a study of gender mortality rates (”age”, ”sex”, ”data”, ”summary”, ”mortality”).

Most importantly, though, is the multitude of topics in which R is clearly being leveraged for its statistical computing capabilities. Topic 12 demonstrates functions for depth-based classification (”ddalpha”, ”points”, ”patterns”, ”depths”, ”classifier”). Topic 55 presents key phrases used in diagnostics for univariate stationary extreme value mixture models such as kernel density estimation (”lambda”, ”kernal”, ”density”, ”kerncentres”, ”gaussian”). Topic 65 shows the use of seeding techniques through random number generation and non-negative matrix factorization (”seed”, ”nmf”, ”rng”, ”set”, ”random”), while topic 75 alludes to the practice of imputing data (”amelia”, ”tcltk”, ”priors”, ”frame”, ”state”). Various statistically-grounded algorithms can be specifically identified in topics as well. For example, topics 84 and 85 demonstrate the use of multi-state Markov models and time series analysis, respectively. Topic 5, 11, and 16 all outline clustering and classification techniques.

When the document entropies of these topics are plotted against those of MATLAB and C++, in Figure 4.1, it can be seen that R topics consistently have a higher document entropy. This demonstrates that in comparison to MATLAB and C++ the topics produced in our modeling, and by extension the applications of R, are more generally spread across packages rather then being highly specific. This is unexpected considering the source code from which R topics were produced had over 9,000 fewer files than the MATLAB source code used in this study in Table 4.1. Further, considering that R is a highly leveraged open-source language that attracts a very diverse community of developers one might expect the application of R to be highly specific rather than generalized. Additionally, we considered the topic uniformity scores assigned to each R topic in comparison to MATLAB and C++ in Figure 4.2. In this comparison, R topics were the

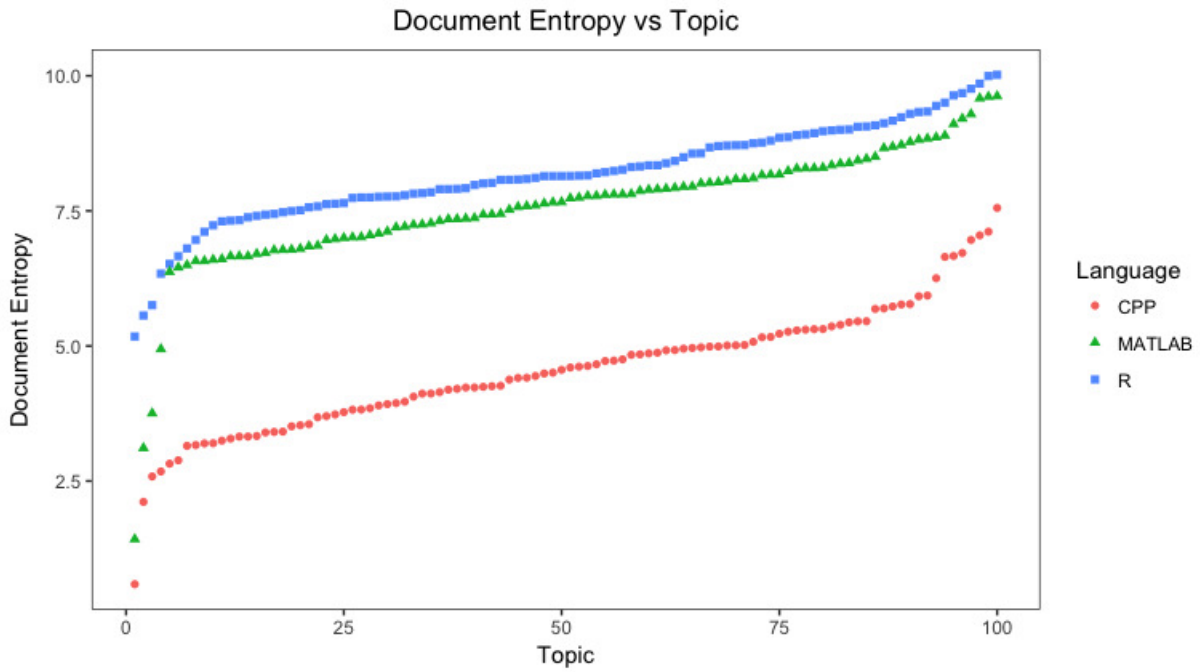


Figure 4.1: This chart shows the document entropy across all topics generated for each of the 3 languages included in this study

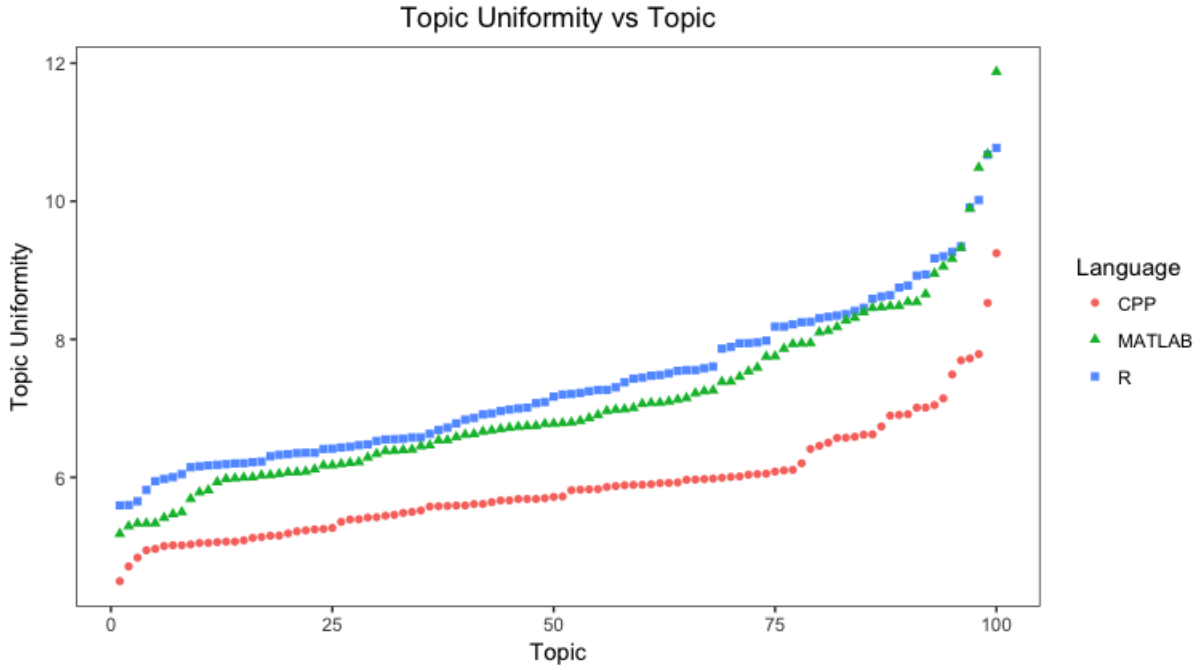


Figure 4.2: This chart shows the uniformity metric across all topics generated for each of the 3 languages included in this study

#	Topic
5	fit family coef model weights
11	cluster dist means result cores
12	ddalpha points patterns depths classifier
13	env gui envir container exists
14	url query api json status
16	species train prediction data lda
26	par lower upper dist distribution
28	gene data verbose ontology flag
32	spec wave frequency seq matrix
34	file read write header append
55	lambda kernel density kerncentres gaussian
59	age sex data summary mortality
65	seed nmf rng set random
75	amelia tcltk priors frame state
84	msm states population transition covariates
85	time date year period series

Table 4.2: A sampling of the 100 topic models created from R source code files.

most uniform across the distribution of topics, indicating that terms tended to co-occur together frequently. This reinforces the trend identified in document entropy. If utilities are more generally applied, producing a high document entropy, their application can be expected to be more uniform. This is the behavior we find demonstrated in the R source code environment.

4.4.2 Topic Modeling: MATLAB Files

The LDA topics represented in Table 4.3 depict a small subset of the 100 topics generated from the MATLAB packages retrieved for this study. Within these topics, the general utilities one would expect in a mature software environment can be found. Topics 11, 33, and 99 demonstrate different graphing applications, while topic 41 encompasses data streaming and topic 50 represents basic data investigation.

Domain-specific applications of MATLAB surface in select topics. These include frequency processing in topic 6, physics-centered computations in topic 40 and 59, as well as numerical computing in topic 67. Topic 77 exhibits functionality within MATLAB associated with EEG and eye tracking experiments. Topic 78 show Other utilities specific to MATLAB appear as well, such as variable-length input arguments implemented by the keyword ”varargin” in topic 17.

More computationally intensive applications of MATLAB are surfaced as well. Topic 2 represents distribution and density calculations through the surfacing of cumulative distribution function, ”cdf”, and probability density function, ”pdf”. Topic 43 covers date-based data analysis in time series (”date”, ”year”, ”day”, ”series”, ”period”). Topic 51 centers on image processing (”image”, ”imshow”, ”gray”, ”imread”, ”rbg”) and topic 95 covers trigonometric calculations (”cos”, ”sin”, ”angle”, ”theta”, ”length”).

Statistically-motivated topics identified demonstrate utilities that differentiate MATLAB from non-statistically leveraged languages. Topic 15 represents cost-sensitive classification (”cost”, ”cluster”, ”prob”, ”size”, ”population”) while topics 24 and 35 cover supervised learning. Topic 60 shows neural network construction (”net”, ”layer”, ”network”, ”hidden”, ”weights”) and topic 13 alludes to leveraging of C++, a practice commonly used to optimize more computationally intensive statistical operations as discussed in Section subsection 4.4.3.

To further our investigation into the application of MATLAB we can again consider all 100 topics generated when compared against those of R and C++ for both document entropy and topic uniformity. In general, the document entropy of MATLAB topics was lower than that of R topics in Figure 4.1. This trend demonstrates that different applications of MATLAB were more specific to smaller groupings of documents rather than being spread across a larger section of source code. Further, MATLAB topics were less uniform than R topics in Figure 4.2 indicating that the application of concepts in MATLAB is less specific compared to R, a surprising trend considering that R is open-source and thus may be leveraged for a more diverse set of problems.

#	Topic
15	cost cluster prob size population
23	node graph tree edges root
27	wavelet qmf filter transform signal
35	class train features labels classifier
40	force velocity stress mass joint
43	daily asset series stock portfolio
51	image imshow gray imread rgb
57	file fprintf filename read fopen
59	heat pressure flow gas density
60	model net layer parameters input
64	font text size alignment style
77	eeg saccade subject eye blink
78	spot indx price call option
92	earth orbit vector gps longitude
95	cos sin angle theta degrees

Table 4.3: A sampling of the 100 topic models created from MATLAB source code files.

4.4.3 Topic Modeling: C++ Files

Within the statistical computing community, the standard approach to implementing particularly computationally intensive programs in R and MATLAB has been to leverage C++. This trend is surfaced in topic modeling conducted on C++ files found within the packages of R and MATLAB. Topic 3 outlines the use of network graphs (”graph”, ”parameters”, ”neighbors”, ”parser”, ”edges”). Topic 5 relates to the implementation of the zmq package [112], used for distributed messaging across platforms and languages (”size”, ”event”, ”mesg”, ”zmq”, ”layer”). Topic 40 features the implementation NOMAD package (Non-smooth Optimization by Mesh Adaptive Direct search) [113] (”nomad”, ”point”, ”eval”, ”mesh”, ”model”).

Aside from these statistically intensive applications general utilities also surface in C++ topic modeling. Topic 14 demonstrates file I/O (”file”, ”string”, ”read”, ”write”, ”stream”), topic 15 deals with matrices (”matrix”, ”row”, ”col”, ”nrow”, ”sum”), and topic 45 demonstrates the basic leveraging of trees which could perhaps allude to deeper statistical leveraging of this data structure (”node”, ”tree”, ”split”, ”parent”, ”child”).

#	Topic
3	graph parameters neighbors parser edges
5	size event mesg zmq layer
14	file string read write stream
15	matrix row col nrow sum
22	mult size kriterium alle genoptions
32	type input traits lhs rhs
40	nomad point eval mesh model
42	license gnu general software version
45	node tree parent child leaf
49	cluster model stk init algo

Table 4.4: A sampling of the topic models created from C++ source code files.

When considering the document entropy and uniformity of C++ topics very clear trends emerge in comparison with R and MATLAB. The document entropy of C++ topics is significantly lower than those of MATLAB and R shown in Figure 4.1. While individual low entropy scores could be due to uncommon coding practices such as topic 32 which implement right-hand side and left-hand side comparison (”type”, ”input”, ”traits”, ”lhs”, ”rhs”) or something more fundamental like comments written in another language as in topic 22 which includes German vocabulary (”mult”, ”size”, ”kriterium”, ”alle”, ”genoptions”). This overall trend demonstrates that while some applications of C++ in statistical computing are more generalized across documents, most are much more specific. This demonstrated specificity confirms the hypothesis that C++ code is embedded in R and MATLAB packages for optimization because the needs of different packages varies, producing much more specific topics. This trend continues when topic uniformity is considered. The uniformity of C++ topics in Figure 4.2 is much lower than MATLAB and R. Since C++ is being used to optimize code this makes sense as the utilities needed for each specific optimization most likely vary package to package.

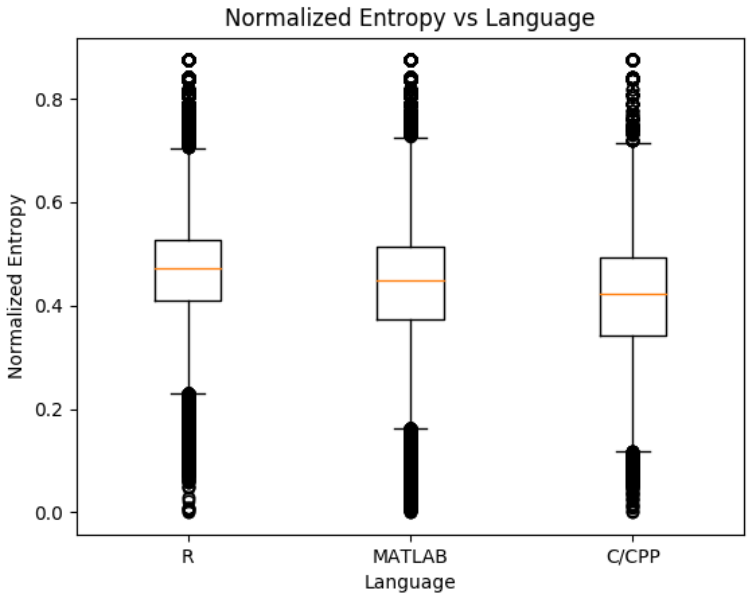


Figure 4.3: This chart shows box and whisker plots of normalized entropy for each language.

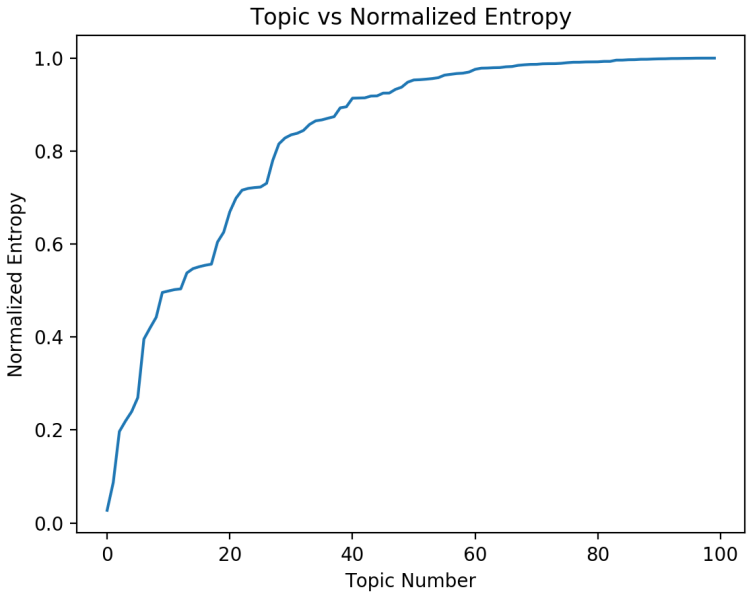


Figure 4.4: The topics shown in Table 4.5 versus their normalized entropies.

4.4.4 Topic Modeling: R, MATLAB, C++

In our analysis of the entire collection of source code, we created LDA topic models of all R and MATLAB source code files combined. In the previous sections, each language was analyzed separately to generate a topic model on only source code of that language. The topic models generated for this combined study can be seen in Table 4.5. We also calculate the relationship between a topic and each of the 2 main languages we are studying (R, MATLAB) by normalizing the summed probability that a specific code file relates to that topic for each language. We then take the entropy of these 2 calculated probabilities and have sorted the topics by this value. A higher entropy then demonstrates that the topic is more widely related to both languages while a lower entropy demonstrates that the topic is more specifically related to a single language, what language that is can be identified by then looking back at the probabilities.

Figure 4.4 illustrates the the normalized entropy across the topics produced, when they are sorted by entropy. The trend in the graph demonstrates that the majority of the topics fall under a high entropy (0.8 or greater). In this context a high entropy is surprising as it indicates that the topic being considered relates in a small amount to a large number of files. Considering the language of R and MATLAB varies, one would expect to find many topics with very low entropies, illustrating the specificity of the languages. However this is not the case, this could indicate that the application of MATLAB and R is more uniform than originally thought. We will investigate the trends of high and low entropy topics in the remainder of this section.

Within the topics generated with low entropies the differences in language between MATLAB and R to implement similar functions is evident. Topic 3 covers the creation of graphs and is highly correlated to MATLAB. This topic is similar in theme to the R correlated topic 12 but is surfaced in a distinct topic due to the use of different keywords across the two languages (”xlab” vs. ”xlabel”, ”ylab” vs. ”ylabel”, etc.). This trend continues in topic 7 which covers the creation of statistical models with keywords like ”input” and ”parameters”. Although this topic is closely related to MATLAB it is similar to the topic 16 which is related to R but contains language that covers testing like ”expect” and ”verbose”.

The R language plays a large role in the open source statistical analysis community. In the topics highly correlated with only R, this is clearly demonstrated. Topics 5 and 8 are related to list manipulation while topic 12 includes common R keywords used when formatting visualizations. Testing keywords like ”test” and ”verbose” appear in topic 16 while data formatting is supported by topic 17 through words like ”format”, ”cat”, and ”round”. There are also less generalized topics found that highlight specific uses of R such as topic 18 which references the use of outside sources through ”url” and ”api” calls as well as the subsequent parsing of the response.

Topics in Table 4.5 highly correlated to MATLAB demonstrate some of the strengths within this language. Aside from the more generalized functions that one would expect to see in a language used widely throughout academia for statistical processing like plotting in topic 3 and model creation in topic 7, there are more specific topics that demonstrate what MATLAB may be utilized for over other languages. MATLAB's ability to handle functions dynamically is represented in topic 2 which holds keywords like “varargin”, a keyword that enables a function to accept any number of input arguments. Other MATLAB specific utilities are also represented as can be seen by image formatting and signal processing topics indicated by topic 11 and topic 13 respectively.

The C++ programming language is commonly leveraged within MATLAB and R packages. This integration is supported by topic 6 and topic 9 which contain keywords like “sexp” and “rcpp”. The use of C++ in R and MATLAB as a way to perform generalized tasks like formatting is demonstrated in topic 4 and topic 15 as well as error coding in topic 20. Other coding best practices also seem to be contained within C++ and then utilized by R and MATLAB. Monitoring code behavior through gsl is shown in topic 19 while versioning of software keywords like “copyright” and “distributed” appear in topic 1.

The normalized entropy across all topics for each language was found to be very similar across each language (R, MATLAB, C++) as demonstrated in Figure 4.3. This is notable as it indicates that no language is particularly unique in its application by the community that uses it, at least in comparison to the other two languages in this study. Further, on a topic by topic basis, when the

topics are sorted based on the normalized entropy in Table 4.5 the top 20 lowest entropies feature MATLAB 6 times, R 7 times, and C++ 7 times. This even representation in low entropy topics, which indicate that one language was significantly more related to that topic than others, further affirms the overall neutrality of the topics produced for this section.

When considering the 10 topics with the highest entropies displayed in Table 4.5 we can extract trends that are generally applicable to all languages. Utilities necessary to a wide range of disciplines surface such as iteration (topic 98), feature extraction and storage (topic 92), and investigating the modality of data (topic 96).

Overall, the topics generated across the entire mined corpus of MATLAB and R source code demonstrate that the application of MATLAB and R, to statistical computing problems, is more uniform than expected, diverging from this uniformity mainly when a difference of keywords comes into play.

Table 4.5: This table shows the topic model, sorted by entropy, created from all MATLAB, R, and C++ source code files. The second column represents how much each language corresponds to that topic. The third column is the normalized entropies of the values in the second column. An entropy of 1 corresponds to the languages being evenly split with the topic. Where as an entropy of 0 means a single language exclusively relates to that topic.

#	R	M	C/C++	Entropy	Topics
1	0.003	0.002	0.993	0.038	license gnu general version software free warranty program copyright org foundation details distributed terms copy modify purpose implied file http
2	0.003	0.993	0.003	0.039	cell size varargin isempty disp fprintf length input nargin zeros strcmp find array numel matlab output repmat sprintf reshape str
3	0.003	0.993	0.003	0.042	plot axis figure title size hold set color subplot xlabel ylabel font grid disp marker gca max legend width zeros
4	0.004	0.006	0.988	0.063	size type array ptr string set begin key back base stream push copy map add cast src element pointer internal
5	0.984	0.007	0.008	0.084	list names length paste stop character call numeric sep missing match lapply fun check levels unique unlist sapply values collapse
6	0.018	0.015	0.966	0.155	sexp integer ret real elt protect set alloc string unprotect pointer logical nil realsxp rprintf length numeric ptr len names
7	0.013	0.958	0.027	0.181	number values output input size order sum points compute zeros parameters linear note functions algorithm point returns space variables matrix
8	0.955	0.016	0.027	0.192	matrix dim length nrow ncol rep sum stop apply list numeric colnames integer cbind seq drop array rownames dimnames max
9	0.030	0.028	0.941	0.240	mat rcpp arma vec numeric trans matrix list named sexp wrap cube cpp pow zeros rcout cal colvec begin rows
10	0.936	0.024	0.039	0.254	code param link item examples eqn number values details numeric seealso description rdname author list logical package title keywords emph
11	0.032	0.934	0.032	0.260	image img size mask images slice pixel max rgb width gray imshow color video min frames pixels height zeros uint
12	0.926	0.039	0.033	0.285	col plot cex xlab par ylab lty lwd ylim type length legend xlim lines pch axis max labels points text

Continued on next page

Table 4.5 – *Continued from previous page*

#	R	M	C/C++	Entropy	Topics
13	0.042	0.922	0.034	0.296	filter signal freq length frequency noise phase aux time fft peak abs wave window max zeros snr amp peaks delay
14	0.057	0.884	0.058	0.399	set position string color style units text callback fig tag axes font parent uicontrol button figure normalized handle visible gui
15	0.060	0.062	0.877	0.415	string status pattern code length set unicode entry format icu len match locale pat start utf text number dest stable
16	0.852	0.084	0.062	0.472	test expect equal verbose check equals identical context tests set expected list equivalent tolerance skip unit works nbr units assert
17	0.838	0.082	0.078	0.503	cat print paste sep results digits round summary cor score format names stats scores output colnames corr items rownames table
18	0.834	0.089	0.076	0.512	url http query api list key json character search org request response session data https message web token www paste
19	0.107	0.104	0.787	0.604	gsl free prop lam rprintf pow matrix malloc calloc ini printf beta set fabs surv time lambda alloc cov log
20	0.088	0.133	0.778	0.616	msg handle assert message tbb device port status task command thread cmd server serial set queue buffer job req connection
21	0.769	0.096	0.133	0.634	res formula family coef terms weights fit response term model call coefficients residuals glm attr intercept data fitted resp offset
22	0.757	0.095	0.146	0.651	test method conf boot level statistic interval sum estimate pval stat surv confidence strata bootstrap risk pvalue values alternative stop
23	0.118	0.748	0.133	0.671	block sys set blocks add system param position time pid blk mdl built step abc gain control input output vel
24	0.131	0.738	0.129	0.687	handles object eventdata set string gui background edit handle guidata color callback matlab future data version reserved fcn structure gcbo
25	0.136	0.139	0.724	0.709	arg git buf bat type len repo mal ctx str oid ptr dst stk free lng commit repository readstat nil
26	0.104	0.177	0.717	0.710	matrix diag sparse inv eigen scalar diagonal matrices svd solve det norm coeff symmetric eig compute real vectors chol product

Continued on next page

Table 4.5 – *Continued from previous page*

#	R	M	C/C++	Entropy	Topics
27	0.075	0.683	0.240	0.726	point cos angle sin points length mesh face radius vertex vertices cgal center coord rad plane direction rot faces rotation
28	0.682	0.169	0.147	0.768	data frame group dataset groups raw meta names variable grp factor variables column values set unique subset columns dat datasets
29	0.641	0.165	0.193	0.819	object method set generic signature methods objects slot type call standard numeric definition list character rdname valid poped show representation
30	0.631	0.214	0.153	0.827	width color plot height panel ggplot text size fill geom scale label top colors aes layout axis colour title legend
31	0.194	0.182	0.623	0.840	graph edge igraph network edges net unit vertex head layer degree nodes ptr tail act vertices directed deg hidden adj
32	0.586	0.297	0.116	0.840	file path dir files filename read output write save txt directory load package csv folder lines run exists sep system
33	0.169	0.618	0.212	0.843	attr mem attributes attribute xxxx tidy att disc core elem subj ida subject mac retval bio matfmatl vode set attrs
34	0.249	0.600	0.150	0.853	flow temperature pressure tmax unit gsw water station wind speed flux density units tmin output fraction depth debug weather velocity
35	0.581	0.169	0.248	0.875	model cov est models fixed parameters var matrix variance list estimate values fit data bic covariance estimates aic covariates estimation
36	0.118	0.372	0.508	0.877	fid file read field header fprintf uint hdr offset write type fread bytes data filename format record len fields fclose
37	0.230	0.194	0.574	0.887	type input args parameter traits rcpp method cpp fun sexp base string lhs rtype rhs result met signature valid ptr
38	0.136	0.340	0.523	0.889	sol bar copyright software conditions provided including binary stri notice tensor limited warranties implied liability source list contributors mpfr disclaimer
39	0.567	0.232	0.199	0.894	pred train predict test model data newdata training prediction set classes auc roc fold predicted loss validation models forest ensemble

Continued on next page

Table 4.5 – *Continued from previous page*

#	R	M	C/C++	Entropy	Topics
40	0.196	0.566	0.236	0.894	cfg channel mutil data channels sint vol chan source time elec isfield trip sens ops label trial errcode freq meg
41	0.145	0.519	0.334	0.898	fix isempty work precision integer dumvar lda complex abs array iwork size real ierr writef kprint format reshape prologue subroutine
42	0.189	0.265	0.545	0.908	index count current max order length indices len left min sort counter total find number flag pair sorted values prev
43	0.235	0.546	0.217	0.912	coef dec ord coefs actual test expctd img subband hex lppufb transform diff wavelet values level dist size zeros sprintf
44	0.208	0.539	0.252	0.916	num str fcn den fcns handle qmx set stdcall ptr task rhs strcat lhs max calltype channel dig reset cstring
45	0.533	0.194	0.272	0.917	gene seq snp chr genes sequence length ids inds ped ann data marker allele gap sequences names annotation fam chrom
46	0.539	0.227	0.232	0.918	fit obs mod number chunk code rnw parm avg data set fitted cand list eval fits summary library type gof
47	0.363	0.165	0.471	0.928	prior mcmc post posterior samples chain beta model burnin iter parms accept thin sample sigma chains log parameters prob gamma
48	0.514	0.243	0.242	0.937	time dat date year times day age start month format dates days years zone posi interval period duration sex calendar
49	0.226	0.511	0.261	0.937	triangle element pts tri boundary energy elements voxel material number field particle point stress dof xdim strain reg dtype basis
50	0.246	0.241	0.511	0.938	row col rows column cols columns rhs lhs glp nrows number ncols matrix mpl csa max check set val add
51	0.500	0.208	0.291	0.939	scale lower upper shape bound length tail bounds log stop list distribution limit eta location numeric link rep prob extra
52	0.269	0.507	0.223	0.939	obj properties set property props methods java cls add access object board prop player objects check component handle update cdata
53	0.491	0.253	0.254	0.951	var env frame variable vars text tkgrid tclvalue paste box command envir sep tcl variables rcmdr sticky entry gettext active
54	0.256	0.253	0.489	0.952	sqlite cache hash page pool fts flags file parse assert key table memory expr cursor mem token list lock free

Continued on next page

Table 4.5 – *Continued from previous page*

#	R	M	C/C++	Entropy	Topics
55	0.218	0.469	0.311	0.956	kernel func reg domain chebfun dom deriv pass bandwidth pref continuous ordered eval ode prod fun norm tol jac unordered
56	0.478	0.272	0.248	0.958	map lat poly spatial area region coords lon raster proj points coordinates geo polygon track polygons bbox grid point projection
57	0.293	0.240	0.466	0.963	lambda digamma knots spline penalty pen string fullcond lambdas datamatrix lasso knot beta intercept degree compute mult back der push
58	0.245	0.288	0.466	0.963	info rank constraints constraint solution problem comm dsdp norm solver set mpi solve dual cone status constr sol objective rhs
59	0.466	0.288	0.245	0.964	diff length max probs min cut fac median breaks quantile arr sum med quant quantiles density abs dens coo perc
60	0.278	0.467	0.254	0.964	val ref rand imp del single reference disp para aux los conj datos details impute las imputation maxtime imputed maxdiff
61	0.276	0.465	0.258	0.965	bin omega hist bins acc json histogram uni contrast hit length nbins contr sum syn sds counts len histo number
62	0.212	0.409	0.378	0.966	par params prob gen pdf distr parameters distribution unur cdf probability set parameter sum check discrete log plate exp distributions
63	0.444	0.234	0.320	0.969	text xml tag html doc word string tags div document content words foo corpus parse character style token css add
64	0.247	0.307	0.444	0.972	base comp bits bit mad ptr token nxs master count string comps symbols phase ctype gas cxx surface msg map
65	0.320	0.241	0.438	0.973	para cross geno covar mixture algo pheno markers marker map qtl dfr strategy chr wts gaussian covars mixmod output criterion
66	0.399	0.376	0.224	0.973	series lag price period trend portfolio time lags forecast date returns stock risk rate prices vol call acf asset trx
67	0.414	0.228	0.357	0.973	tree root species node edge split child parent tip phy length branch children phylo leaf nodes trees leaves list taxa
68	0.256	0.436	0.307	0.977	des def vect fin nom les traj coord pour pro fprintf indice est pas orbit matrice dans sun quali mog

Continued on next page

Table 4.5 – *Continued from previous page*

#	R	M	C/C++	Entropy	Topics
69	0.437	0.299	0.263	0.977	sim rate site eval simulation process list set nsim rates sites echo simulate fig type simulated cap expression length simulations
70	0.427	0.253	0.319	0.979	table sql select rel tbl list tables conn query create column join schema type dplyr categories category connection database res
71	0.266	0.306	0.427	0.981	node idx nodes arc graph links ctrl parents parent dag arcs garch list modelinc set find max spec ipars dfa
72	0.340	0.401	0.258	0.985	options threshold opts option thresh check length thr thresholds outliers kmax method outlier robust flag set gpd phiu sde center
73	0.417	0.292	0.289	0.985	theta sigma tau rho sqrt copula der teta log exp sum length yych cop eta thetas pdf family margins derp
74	0.380	0.365	0.253	0.986	range curve dose calc band max interp min values trim curves length fct sat spectra ranges wavelength spct bands lum
75	0.403	0.264	0.332	0.986	dist cluster distance weight weights clusters sum clust parallel method matrix number distances weighted min metric max centers clustering size
76	0.408	0.286	0.305	0.988	gtk window widget type text container handler action view check gdk button add set ptr svalue gui icon package call
77	0.392	0.334	0.272	0.990	label labels basis smooth lab proc fuzzy data list functional sse values meth smoothing fdata nclass scores coefs length nbasis
78	0.401	0.313	0.285	0.990	beta delta phi gamma lambda eta psi hat sum sqrt kappa omega star inv alpha diag abs solve exp alfa
79	0.398	0.319	0.282	0.990	sample pars samples power cost size samp trace stage sampling length number sizes sam traces nmax replace seismic costs total
80	0.395	0.278	0.326	0.990	log alpha exp expr sum lik likelihood sqrt loglik lgamma ifelse alphas mle length numeric dep exponential distribution grad sealen
81	0.385	0.275	0.338	0.991	random seed rand profile slope rng mass prec dummy number generate runif length generator reps nmf rnorm dum ran profiles
82	0.305	0.298	0.395	0.992	result user err ptr object check cutoff type version pointer gint file atk cdr data cancellable setcar info callback string

Continued on next page

Table 4.5 – *Continued from previous page*

#	R	M	C/C++	Entropy	Topics
83	0.363	0.273	0.362	0.992	temp loc pop curr locus swf loci length allele alleles sum freq hap genotype number freqs populations rare genotypes population
84	0.334	0.384	0.280	0.992	con stack summ decision imax rst imin glob jmax relation max cue ladder nodo truth tee del arrival rapidjson nominal
85	0.286	0.324	0.388	0.992	state rule states buffer depth transition cimg rules gram tth setup dparser lex scanner current trans scan bus arg transitions
86	0.285	0.332	0.381	0.993	config population pop batch crit size spot alg objective fitness shark number algorithm criteria set multi funstr parameters individual search
87	0.305	0.386	0.307	0.994	ind ext color xyz rgb coord red texture green extproc lenum init arb vertex apientry tex lint blue thres lab
88	0.384	0.297	0.317	0.994	design factor effect cond effects eff mix treat treatment outcome lik trt gpcf fact factors study condition marg sum full
89	0.384	0.308	0.306	0.994	sig vals perm icd counts dif permutation chi permutations cases miss itr res focal sum controls major permute spp nperm
90	0.316	0.300	0.383	0.994	vec spec span zeta history length blpapi order quad delt add tst checkb max tvec lamb print set fnu xvec
91	0.380	0.324	0.294	0.994	tab sel xmax seg xmin ymax measure comb ymin gvar biplot smat segs min transf tabs points mds gbp river
92	0.308	0.314	0.376	0.996	level feature alt storage trees stages features lst bson opencv haarcascade frontalface mongoc array msa val feat bmp imread mongo
93	0.353	0.353	0.293	0.996	target event source events trial sgp time baseline times dose iso targets eeg trials number grade epoch sources conc subject
94	0.295	0.343	0.361	0.996	roi run historical scenario data list rcp amon tas air spm pcs rcm reloclim amount cmip file update esm gfdl
95	0.343	0.299	0.357	0.997	sym segment mark segments marks parser rmax npoints npts window balance yyval bws yyvsp correction points ppp ppm action xrange
96	0.330	0.306	0.362	0.997	tmp start mode modes inp starts points leb shimmer ends gauss starting stop lebedev reccurence neuron xtmp output bhat mvar

Continued on next page

Table 4.5 – *Continued from previous page*

#	R	M	C/C++	Entropy	Topics
97	0.356	0.332	0.311	0.998	pos grid cnt neg make desc dev learner positions position task card length regr mydata ngrid hand mumps tiling asym
98	0.321	0.356	0.322	0.998	max min iter control opt step tol init eps abs method initial grad iterations fun norm optim gradient conv iteration
99	0.325	0.349	0.324	0.999	stat cont part length mid big joint alp beam cairo width partition pad spacing pin spc ndim funwords parts moment
100	0.337	0.324	0.338	0.999	cur low high rec symbol cum atom ctr matrix upp locs spike length lab spikes atoms support ntimes scl step

4.5 Related Works

While we believe we are the first to take a machine learning approach to analyzing R and MATLAB with topic models, including a preliminary topic analysis [114], previous efforts have been made to gain insight into the R ecosystem. Others have analyzed the strengths and weaknesses of R, such as Caragea et al. in their SWOT (Strengths, Weaknesses, Opportunities, Threats) analysis [115], or Culpepper et al. in their review of the limitations and benefits of a statistical computing tool that is continuously being updated [116]. German et al. inspected R packages found on CRAN, although they focused specifically on the growth and evolution of core and contributed projects over time. This work was extended to other repositories in [117], which analyzed the origin and dependencies SoftwareMining-2017, October 2017, Urbana-Champaign, Illinois USA of more than 12,000 packages in order to more completely characterize the R ecosystem. Despite the inclusion of more data, they concluded that CRAN still remained central to the R ecosystem, although other repositories, mainly GitHub, were being used to leverage the advantages that accompany open source development.

Some have taken a different approach to analyzing the R packages and other resources that can be downloaded from CRAN. The work in [118] inspected the large range of style guides and naming conventions that were in use throughout packages pulled from CRAN, and noted many inconsistencies. These inconsistencies led, in part, to our own decision to not do a deep parse of R code for this pilot study and instead only focus on API documentation.

Others have noted the rise of the R programming language and what it could mean for statistical computing. Hornick, 10 years after his original analysis of R in [119], which largely introduced the language to the statistical community, returned to inspect the evolution of R through the lens of the multitude of R packages that had become available in the intervening decade [120]. This later work served to encourage the statistical computing community to work towards establishing a common understanding of software quality.

After a literature review, we noticed a significant gap in research pertaining to the MATLAB environment. Many papers noted above study the R ecosystem, but few MATLAB equivalents were found. The Mathworks team claims the simplicity of the MATLAB language and its similarity to English, make it easier to learn compared to R which was created for statisticians [121]. They also report that MATLAB tends to be faster than R for technical computing tasks, statistics, and machine learning in part due to built in multi-threading. The authors in [122], performed aspect mining and studied the cross cutting concerns showing the advantages of aspect oriented programming in MATLAB.

Though research of the MATLAB ecosystem is limited, other programming languages have been studied in informative ways. Linstead et al. also extracted concerns from Java source code, as latent topics, to study the Java software vocabulary and measured scattering and tangling of those topics [123]. Zhang et al. [124] also explore topics present in Java source code in several projects chosen from the large corpus curated in [125]. The authors propose a new topic extraction method, Embedded Topic Extraction (EmbTE), and compare the topics found to those produced LDA and Non-negative Matrix Factorization (NMF). Perez et al analyzes and compares Python to R and other established computing languages in [126]. They note the rapid progress of Python

numerical processing, documentation, data visualization, and the language's other high-quality tools. Ugurel et al. mined source code of multiple languages from IBiblio, Sourceforge, as well as other archives in a classification study [127]. Though not topic modeling, they performed feature extraction through expected entropy loss. The authors trained a support vector machine to classify the source code into eleven application topics and ten programming languages based on those features. Although topic classification was only attempted on C/C++ source files, the language classification analysis consisted of eleven categories including MATLAB. The authors in [128], expanded on this study to include API calls from third party libraries as possible features for topic classification in closed and open source Java applications. The authors in [129], take a deep dive into several LDA hyper-parameters and their impact on the resulting LDA model.

With precedent established for other languages and a lack in research pertaining to MATLAB environment, we believe this could be a significant future research area. We are confident in the assumption that R and MATLAB's influence in the world of computing will continue to grow in the coming years. With this growth, it is imperative to continue to investigate these languages, and similar languages, to understand fundamental differences from the procedural and object-oriented technologies that have been the emphasis of research in mining software repositories. To this end, we intend to expand the initial work described here to include more facets of contemporary scientific computing software packages. To start, we will expand our corpus by parsing the original R source code in addition to the RPM documentation. This will allow us to study facets of R programming at a lower level of granularity. Additionally, we would like to explore the naming conventions of R, and how they compare to what has already been observed in languages such as Java [123].

5 Conclusion

This dissertation has highlighted three applications of machine learning in previously unexplored and under explored areas. A central theme of these applications has been applying and enabling machine learning use in new domains.

In Chapter 2, we extended previous work regarding the application of machine learning techniques for classification of UML images. Transfer learning allows us to take, in effect, a shortcut in training deep architectures. Given limited data, it is nearly impossible to train a network with the depth and substantial number of parameters as in VGG. However, by transferring knowledge learned from one task to another, we are able to tune off-the-shelf deep architectures and achieve high classification accuracy, rather than having to design new architectures with fewer layers and smaller parameter spaces to learn. Most importantly, the knowledge that forms the basis of the transfer learning needs no previous exposure to artifacts from the software domain, suggesting that transfer learning can be applied broadly to applications of deep learning within empirical software engineering.

Experimental results have shown training is positively effected by transfer learning even when the number of samples shown to the network is kept small. In contrast, even a smaller network with substantially fewer parameters is unable to learn as well. As a control, an off-the-shelf VGG network was also tested and the entire architecture containing over 14 million parameters was allowed to train. As expected, this network failed to improve beyond 50% accuracy even when shown the maximum number of samples tested.

In addition to affirming the utility of transfer learning for mining software artifacts, our results suggest that as a research community we should be more proactive in curating machine learning models trained on software data, in addition to the software data itself. Such repositories of pre-trained models would allow empirical software engineering researchers to apply transfer learning to new applications using models already tuned using software data of various types.

The ubiquitousness of deep learning has resulted from extensive free and open source libraries [49, 50, 51]. Deep learning's success and popularity merit its integration in large-scale computing packages, like those written in Fortran. Instead of rewriting all existing libraries in Fortran, we introduced a two-way bridge between low-level, Fortran, and Python through the FKB Library in Chapter 3. The library provides researchers the ability to implement neural networks into Fortran code bases while being able to transfer them back and forth with Keras.

Fortran, which has been a staple within computationally intensive fields for decades, will undoubtedly see continued use due to its fast computational ability and vast amounts of legacy code. The FKB library enables users to access many features of the Keras API directly in Fortran, including the ability to create custom layers and loss functions to suit their needs. We demonstrate the integrability of FKB through our case study involving the SPCAM3 simulator. An advantage of FKB is its ease of use, demonstrated by its ability to be compiled in advance and once linked can be easily leveraged in existing large scale simulators, as we have illustrated for the application of multi-scale physical simulations of the global atmosphere. In Chapter 4, the use of Latent Dirichlet Allocation allowed insight into specific use cases and application domains of the statistical programming languages analyzed in this study. LDA models of individual languages allow for analysis of topics, specific to that programming paradigm. Topic models of all languages allow cross-cutting concerns to be identified, while quantitatively analyzing (entropy) how much a topic relates to all languages. C++ files, used in conjunction with R and MATLAB, are specifically for optimization tasks. This is confirmed by C++ topics focusing on high efficiency and optimization. Additionally, low entropy and high uniformity scores signal highly specific use cases. Topics gathered from R and MATLAB models show a plethora of statistical computing

capabilities. This non-surprising result attests to the validity of methods used in this study. Our results indicate that MATLAB is used for more specific domains. This is confirmed by MATLAB's lower document entropy score compared to R, as well as a higher uniformity score. The results of our analysis provide insight into the underlying characteristics of scientific computing algorithms in general, with potential to guide further research in this area.

REFERENCES

[1] D. Hubel and T. Wiesel, "Receptive fields, binocular interaction and functional architecture in the cat's visual cortex," *The Journal of Physiology*, vol. 160, 1962.

[2] J. Ott, E. Linstead, N. LaHaye, and P. Baldi, "Learning in the machine: To share or not to share?" *Neural Networks*, vol. 126, pp. 235–249, 2020.

[3] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *ICML*, 2010, pp. 807–814.

[4] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[5] B. Zhou, A. Khosla, À. Lapedriza, A. Oliva, and A. Torralba, "Learning deep features for discriminative localization," *CoRR*, vol. abs/1512.04150, 2015. arXiv: 1512.04150.

[6] R. Kotikalapudi and contributors, *Keras-vis*, https://github.com/raghakot/keras-vis, 2017.

[7] S. J. Pan and Q. Yang, "A survey on transfer learning," *IEEE Transactions on Knowledge and Data Engineering*, vol. 22, no. 10, pp. 1345–1359, Oct. 2010.

[8] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A Large-Scale Hierarchical Image Database," in *CVPR09*, 2009.

[9] J. Ott, A. Atchison, and E. J. Linstead, "Exploring the applicability of low-shot learning in mining software repositories," *Journal of Big Data*, vol. 6, no. 1, p. 35, May 2019.

[10] E. S. Olivas, J. D. M. Guerrero, M. M. Sober, J. R. M. Benedito, and A. J. S. Lopez, *Handbook Of Research On Machine Learning Applications and Trends: Algorithms, Methods and Techniques - 2 Volumes*. Hershey, PA: Information Science Reference - Imprint of: IGI Publishing, 2009, ISBN: 1605667668.

[11] D. M. Blei, A. Y. Ng, and M. I. Jordan, "Latent dirichlet allocation," *Journal of Machine Learning Research*, vol. 3, no. Jan, pp. 993–1022, 2003.

[12] H. M. Wallach, D. M. Mimno, and A. McCallum, "Rethinking lda: Why priors matter," in *Advances in neural information processing systems*, 2009, pp. 1973–1981.

[13] J. Ott, A. Atchison, P. Harnack, A. Bergh, and E. Linstead, "A deep learning approach to identifying source code in images and video," May 2018, pp. 376–386.

[14] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, *et al.*, "Imagenet large scale visual recognition challenge," *International journal of computer vision*, vol. 115, no. 3, pp. 211–252, 2015.

[15] K. Simonyan and A. Zisserman, *Very deep convolutional networks for large-scale image recognition*, 2014. arXiv: 1409.1556 [cs.CV].

[16] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, ser. NIPS'12, Lake Tahoe, Nevada: Curran Associates Inc., 2012, pp. 1097–1105.

[17] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *CoRR*, vol. abs/1512.03385, 2015. arXiv: 1512.03385.

[18] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Computer Vision and Pattern Recognition (CVPR)*, 2015.

[19] H. Shin, H. R. Roth, M. Gao, L. Lu, Z. Xu, I. Nogues, J. Yao, D. Mollura, and R. M. Summers, "Deep convolutional neural networks for computer-aided detection: Cnn architectures, dataset characteristics and transfer learning," *IEEE Transactions on Medical Imaging*, vol. 35, no. 5, pp. 1285–1298, May 2016.

[20] N. Bayramoglu and J. Heikkilä, "Transfer learning for cell nuclei classification in histopathology images," in *Computer Vision – ECCV 2016 Workshops*, G. Hua and H. Jégou, Eds., Cham: Springer International Publishing, 2016, pp. 532–539, ISBN: 978-3-319-49409-8.

[21] J. Ott, A. Atchison, P. Harnack, N. Best, H. Anderson, C. Firmani, and E. Linstead, "Learning lexical features of programming languages from imagery using convolutional neural networks," in *Proceedings of the 26th Conference on Program Comprehension*, ACM, 2018, pp. 336–339.

[22] M. Alahmadi, J. Hassel, B. Parajuli, S. Haiduc, and P. Kumar, "Accurately predicting the location of code fragments in programming video tutorials using deep learning," in *Proceedings of the 14th International Conference on Predictive Models and Data Analytics in Software Engineering*, ACM, 2018, pp. 2–11.

[23] R. Hebig, T. H. Quang, M. R. V. Chaudron, G. Robles, and M. A. Fernandez, "The quest for open source projects that use uml: Mining github," in *Proceedings of the ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems*, ser. MODELS '16, Saint-malo, France: ACM, 2016, pp. 173–183, ISBN: 978-1-4503-4321-3.

[24] Y. Bengio, "Practical recommendations for gradient-based training of deep architectures," *CoRR*, vol. abs/1206.5533, 2012. arXiv: 1206.5533.

[25] T. Ho-Quang, M. R. V. Chaudron, I. Samuelsson, J. Hjaltason, B. Karasneh, and H. Osman, "Automatic classification of uml class diagrams from images," *2014 21st Asia-Pacific Software Engineering Conference*, vol. 1, pp. 399–406, 2014.

[26] J. Hjaltason and I. Samuelsson, "Automatic classification of uml class diagrams through image feature extraction and machine learning," 2015.

[27] V. Moreno, G. Génova, M. Alejandres, and A. Fraga, "Automatic classification of web images as uml diagrams," in *Proceedings of the 4th Spanish Conference on Information Retrieval*, ser. CERI '16, Granada, Spain: ACM, 2016, 17:1–17:8, ISBN: 978-1-4503-4141-7.

[28] R. Krishna and T. Menzies, "Bellwethers: A Baseline Method For Transfer Learning," *arXiv e-prints*, arXiv:1703.06218, arXiv:1703.06218, Mar. 2017. arXiv: 1703.06218 [cs.SE].

[29] E. Kocaguneli, T. Menzies, and E. Mendes, "Transfer learning in effort estimation," *Empirical Softw. Engg.*, vol. 20, no. 3, pp. 813–843, Jun. 2015.

[30] Y. Ma, G. Luo, X. Zeng, and A. Chen, "Transfer learning for cross-company software defect prediction," *Inf. Softw. Technol.*, vol. 54, no. 3, pp. 248–256, Mar. 2012.

[31] X. Jing, F. Wu, D. Xiwei, F. qi, and B. Xu, "Heterogeneous cross-company defect prediction by unified metric representation and cca-based transfer learning," Aug. 2015, pp. 496–507.

[32] P. Jamshidi, M. Velez, C. Kästner, N. Siegmund, and P. Kawthekar, "Transfer learning for improving model predictions in highly configurable software," *CoRR*, vol. abs/1704.00234, 2017. arXiv: 1704.00234.

[33] N. Tajbakhsh, J. Shin, S. Gurudu, R. Hurst, C. Kendall, M. Gotway, and J. Liang, "Convolutional neural networks for medical image analysis: Full training or fine tuning?" *IEEE Transactions on Medical Imaging*, vol. 35, no. 5, pp. 1299–1312, May 2016.

[34] IBM, *Fortran*, https://www.ibm.com/ibm/history/ibm100/us/en/icons/fortran/, Mar. 2011.

[35] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[36] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural Networks*, vol. 61, pp. 85–117, 2015.

[37] X. X. Zhu, D. Tuia, L. Mou, G.-S. Xia, L. Zhang, F. Xu, and F. Fraundorfer, "Deep learning in remote sensing: A comprehensive review and list of resources," *IEEE Geoscience and Remote Sensing Magazine*, vol. 5, no. 4, pp. 8–36, 2017.

[38] N. LaHaye, J. Ott, M. J. Garay, H. M. El-Askary, and E. Linstead, "Multi-modal object tracking and image fusion with unsupervised deep learning," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 12, no. 8, pp. 3056–3066, 2019.

[39] J. Ott, A. Atchison, P. Harnack, A. Bergh, and E. Linstead, "A deep learning approach to identifying source code in images and video," in *2018 IEEE/ACM 15th International Conference on Mining Software Repositories (MSR)*, IEEE, 2018, pp. 376–386.

[40] J. Tompson, M. Stein, Y. Lecun, and K. Perlin, "Real-time continuous pose recovery of human hands using convolutional networks," *ACM Transactions on Graphics (ToG)*, vol. 33, no. 5, p. 169, 2014.

[41] J. Ott, A. Atchison, and E. J. Linstead, "Exploring the applicability of low-shot learning in mining software repositories," *Journal of Big Data*, vol. 6, no. 1, p. 35, 2019.

[42] G. Urban, P. Tripathi, T. Alkayali, M. Mittal, F. Jalali, W. Karnes, and P. Baldi, "Deep Learning Achieves near Human-level Polyp Detection in Screening Colonoscopy," *Gastroenterology*, vol. 155, no. 4, pp. 1069–1078, 2018.

[43] F. Agostinelli, S. McAleer, A. Shmakov, and P. Baldi, "Solving the rubik's cube with deep reinforcement learning and search," *Nature Machine Intelligence*, vol. 1, no. 8, pp. 356–363, 2019.

[44] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, *et al.*, "Mastering the game of go with deep neural networks and tree search," *nature*, vol. 529, no. 7587, p. 484, 2016.

[45] M. Raissi, P. Perdikaris, and G. E. Karniadakis, "Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations," *Journal of Computational Physics*, vol. 378, pp. 686–707, 2019.

[46] T. Beucler, M. Pritchard, S. Rasp, P. Gentine, J. Ott, and P. Baldi, "Enforcing analytic constraints in neural-networks emulating physical systems," *arXiv preprint arXiv:1909.00912*, 2020.

[47] Y. Bar-Sinai, S. Hoyer, J. Hickey, and M. P. Brenner, "Learning data-driven discretizations for partial differential equations," *Proceedings of the National Academy of Sciences*, vol. 116, no. 31, pp. 15 344–15 349, 2019.

[48] S. H. Rudy, S. L. Brunton, J. L. Proctor, and J. N. Kutz, "Data-driven discovery of partial differential equations," *Science Advances*, vol. 3, no. 4, e1602614, 2017.

[49] F. Chollet *et al.*, *Keras*, https://github.com/fchollet/keras, 2015.

[50] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, *et al.*, "Tensorflow: A system for large-scale machine learning," in *12th Symposium on Operating Systems Design and Implementation 2016)*, 2016, pp. 265–283.

[51] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in pytorch," 2017.

[52] J. W. Hurrell, M. M. Holland, P. R. Gent, S. Ghan, J. E. Kay, P. J. Kushner, J.-F. Lamarque, W. G. Large, D. Lawrence, K. Lindsay, *et al.*, "The community earth system model: A framework for collaborative research," *Bulletin of the American Meteorological Society*, vol. 94, no. 9, pp. 1339–1360, 2013.

[53] J.-C. Golaz, P. M. Caldwell, L. P. Van Roekel, M. R. Petersen, Q. Tang, J. D. Wolfe, G. Abeshu, V. Anantharaj, X. S. Asay-Davis, D. C. Bader, *et al.*, "The doe e3sm coupled model version 1: Overview and evaluation at standard resolution," *Journal of Advances in Modeling Earth Systems*, vol. 11, no. 7, pp. 2089–2129, 2019.

[54] I. Held, H. Guo, A. Adcroft, J. Dunne, L. Horowitz, J. Krasting, E. Shevliakova, M. Winton, M. Zhao, M. Bushuk, *et al.*, "Structure and performance of gfdl's cm4. 0 climate model," *Journal of Advances in Modeling Earth Systems*, vol. 11, no. 11, pp. 3691–3727, 2019.

[55] N. S. Team, *Nemo ocean engine*, Scientific Notes of Climate Modelling Center 27, Zenodo.

[56] A. Wallcraft, H. Hurlburt, E. J. Metzger, E. Chassignet, J. Cummings, and O. M. Smedstad, "Global ocean prediction using hycom," in *2007 DoD High Performance Computing Modernization Program Users Group Conference*, 2007, pp. 259–262.

[57] M. A. Donelan, M. Curcic, S. S. Chen, and A. K. Magnusson, "Modeling waves and wind stress," *Journal of Geophysical Research: Oceans*, vol. 117, no. C11, 2012.

[58] J. G. Powers, J. B. Klemp, W. C. Skamarock, C. A. Davis, J. Dudhia, D. O. Gill, J. L. Coen, D. J. Gochis, R. Ahmadov, S. E. Peckham, G. A. Grell, J. Michalakes, S. Trahan, S. G. Benjamin, C. R. Alexander, G. J. Dimego, W. Wang, C. S. Schwartz, G. S. Romine, Z. Liu, C. Snyder, F. Chen, M. J. Barlage, W. Yu, M. G., and Duda, "The weather research and forecasting model: Overview, system efforts, and future directions," *Bulletin of the American Meteorological Society*, vol. 98, no. 8, pp. 1717–1737, 2017.

[59] E. Madenci and I. Guven, *The finite element method and applications in engineering using ANSYS®*. Springer, 2015.

[60] L. Börgesson, "Abaqus," in *Developments in geotechnical engineering*, vol. 79, Elsevier, 1996, pp. 565–570.

[61] Y. D. Murray *et al.*, "Users manual for ls-dyna concrete material model 159," United States. Federal Highway Administration. Office of Research …, Tech. Rep., 2007.

[62] D. Komatitsch, J.-P. Vilotte, J. Tromp, J.-P. Ampuero, K. Bai, P. Basini, C. Blitz, E. Bozdag, E. Casarotti, J. Charles, M. Chen, P. Galvez, D. Goddeke, V. Hjorleifsdottir, J. Labarta, N. Le Goff, P. Le Loher, M. Lefebvre, Q. Liu, Y. Luo, A. Maggi, F. Magnoni, R. Martin, R. Matzen, D. McRitchie, M. Meschede, P. Messmer, D. Michea, S. Nadh Somala, T. Nissen-Meyer, D. Peter, M. Rietmann, E. de Andrade, B. Savage, B. Schuberth, A. Sieminski, L. Strand, C. Tape, Z. Xie, and H. Zhu, *Specfem3d cartesian v2.0.2 [software]*, Computational Infrastructure for Geodynamics, 2012.

[63] F. Archambeau, N. Méchitoua, and M. Sakiz, "Code Saturne: A Finite Volume Code for the computation of turbulent incompressible flows - Industrial Applications," *International Journal on Finite Volumes*, vol. 1, no. 1, http://www.latp.univ–mrs.fr/IJFV/spip.php?article3, Feb. 2004.

[64] J. W. L. Paul F. Fischer and S. G. Kerkemeier, *nek5000 Web page*, http://nek5000.mcs.anl.gov, 2008.

[65] B. Brooks, R. Bruccoleri, B. Olafson, D. States, S. Swaminathan, and M. Karplus, "Charmm: A program for macromolecular energy, minimization, and dynamics calculations," *Journal of Computational Chemistry*, vol. 4, pp. 187–217, Sep. 2004.

[66] M. Valiev, E. J. Bylaska, N. Govind, K. Kowalski, T. P. Straatsma, H. J. J. Van Dam, D. Wang, J. Nieplocha, E. Apra, T. L. Windus, and W. A. de Jong, "Nwchem: A comprehensive and scalable open-source solution for large scale molecular simulations," *Computer Physics Communications*, vol. 181, no. 9, pp. 1477–1489, 2010.

[67] A. Ferrari, P. Sala, A. Fasso, and J. Ranft, "Fluka: A multi-particle transport code," *CERN Yellow report*, vol. 2005-10, Jan. 2005.

[68] G.-Q. Jiang, J. Xu, and J. Wei, "A deep learning algorithm of neural network for the parameterization of typhoon-ocean feedback in typhoon forecast models," *Geophysical Research Letters*, vol. 45, no. 8, pp. 3706–3716, 2018. eprint: https://agupubs.onlinelibrary.wiley.com/doi/pdf/10.1002/2018GL077004.

[69] J. Ling, A. Kurzawski, and J. Templeton, "Reynolds averaged turbulence modelling using deep neural networks with embedded invariance," *Journal of Fluid Mechanics*, vol. 807, pp. 155–166, 2016.

[70] H. R. Vega-Carrillo, V. M. Hernández-Dávila, E. Manzanares-Acuña, G. A. Mercado, E. Gallego, A. Lorente, W. A. Perales-Muñoz, and J. A. Robles-Rodríguez, "Artificial neural networks in neutron dosimetry," *Radiation Protection Dosimetry*, vol. 118, no. 3, pp. 251–259, Oct. 2005. eprint: https://academic.oup.com/rpd/article-pdf/118/3/251/4543434/nci354.pdf.

[71] S. Rasp, M. S. Pritchard, and P. Gentine, "Deep learning to represent subgrid processes in climate models," *Proceedings of the National Academy of Sciences*, vol. 115, no. 39, pp. 9684–9689, 2018.

[72] N. D. Brenowitz and C. S. Bretherton, "Prognostic validation of a neural network unified physics parameterization," *Geophysical Research Letters*, vol. 45, no. 12, pp. 6289–6298, 2018.

[73] Kaggle, *2018 kaggle ml & ds survey*, 2018.

[74] Kaggle, *2019 state of data science and machine learning*, 2019.

[75] M. Curcic, "A parallel fortran framework for neural networks and deep learning," in *ACM SIGPLAN Fortran Forum*, ACM, vol. 38, 2019, pp. 4–21.

[76] J. Bernal, "Neurbt: A program for computing neural networks for classification using batch learning," Feb. 2015.

[77] J. Bernal and J. Torres-Jimenez, "Sagrad: A program for neural network training with simulated annealing and the conjugate gradient method," *Journal of research of the National Institute of Standards and Technology*, vol. 120, p. 113, Jun. 2015.

[78] P. Brierley, *Fortran90 mlp backprop code*, http://www.philbrierley.com/phil.html.

[79] S. Nissen, "Implementation of a fast artificial neural network library (fann)," Dec. 2003.

[80] D. Lary, M. Müller, and H. Mussa, "Using neural networks to describe tracer correlations," *Atmospheric Chemistry and Physics*, vol. 4, no. 1, pp. 143–146, 2004.

[81] L. Hertel, J. Collado, P. Sadowski, J. Ott, and P. Baldi, "Sherpa: Robust hyperparameter optimization for machine learning," *Submitted to SoftwareX*, 2020.

[82] J. Snoek, H. Larochelle, and R. P. Adams, "Practical bayesian optimization of machine learning algorithms," in *Advances in neural information processing systems*, 2012, pp. 2951–2959.

[83] J. Bergstra, D. Yamins, and D. D. Cox, "Hyperopt: A python library for optimizing the hyperparameters of machine learning algorithms," in *Proceedings of the 12th Python in science conference*, Citeseer, 2013, pp. 13–20.

[84] D. J. Gagne, T. McCandless, B. Kosovic, A. DeCastro, R. Loft, S. E. Haupt, and B. Yang, "Machine learning parameterization of the surface layer: Bridging the observation-modeling gap," *AGUFM*, vol. 2019, IN44A–04, 2019.

[85] D. J. Gagne, C.-C. Chen, and A. Gettelman, "Emulation of bin microphysical processes with machine learning," in *100th American Meteorological Society Annual Meeting*, AMS, 2020.

[86] N. D. Brenowitz, T. Beucler, M. Pritchard, and C. S. Bretherton, *Interpreting and stabilizing machine-learning parametrizations of convection*, 2020. arXiv: 2003.06549 [physics.ao-ph].

[87] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[88] P. Baldi and P. Sadowski, "The dropout learning algorithm," *Artificial intelligence*, vol. 210, pp. 78–122, 2014.

[89] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *arXiv preprint arXiv:1502.03167*, 2015.

[90] F. Chollet, *Deep Learning mit Python und Keras: Das Praxis-Handbuch vom Entwickler der Keras-Bibliothek*. MITP-Verlags GmbH & Co. KG, 2018.

[91] W. W. Grabowski, "Coupling cloud processes with the large-scale dynamics using the cloud-resolving convection paramaterization (CRCP)," *Journal of the Atmospheric Sciences*, vol. 58, no. 9, pp. 978–997, 2001.

[92] M. Khairoutdinov, D. Randall, and C. DeMott, "Simulations of the atmospheric general circulation using a cloud-resolving model as a superparameterization of physical processes," *Journal of the Atmospheric Sciences*, vol. 62, no. 7, pp. 2136–2154, 2005.

[93] M. Khairoutdinov, C. DeMott, and D. Randall, "Evaluation of the simulated interannual and subseasonal variability in an amip-style simulation using the csu multiscale modeling framework," *Journal of Climate*, vol. 21, no. 3, pp. 413–431, 2008.

[94] K. Thayer-Calder and D. A. Randall, "The role of convective moistening in the madden–julian oscillation," *Journal of the Atmospheric Sciences*, vol. 66, no. 11, pp. 3297–3312, 2009.

[95] S. Rasp, "Online learning as a way to tackle instabilities and biases in neural network parameterizations," *arXiv preprint arXiv:1907.01351*, 2019.

[96] D. J. Gagne II, H. M. Christensen, A. C. Subramanian, and A. H. Monahan, "Machine learning for stochastic parameterization: Generative adversarial networks in the lorenz '96 model," *Journal of Advances in Modeling Earth Systems*, vol. 12, no. 3, e2019MS001896, 2020, e2019MS001896 10.1029/2019MS001896. eprint: https://agupubs.onlinelibrary.wiley.com/doi/pdf/10.1029/2019MS001896.

[97] M. S. Pritchard, C. S. Bretherton, and C. A. DeMott, "Restricting 32–128 km horizontal scales hardly affects the mjo in the superparameterized community atmosphere model v. 3.0 but the number of cloud-resolving grid columns constrains vertical mixing," *Journal of Advances in Modeling Earth Systems*, vol. 6, no. 3, pp. 723–739, 2014.

[98] P. Gentine, M. Pritchard, S. Rasp, G. Reinaudi, and G. Yacalis, "Could machine learning break the convection parameterization deadlock?" *Geophysical Research Letters*, vol. 45, no. 11, pp. 5742–5751, 2018.

[99] MATLAB, *version 9.2.0 (R2017b)*. Natick, Massachusetts: The MathWorks Inc., 2017.

[100] R Development Core Team, *R: A language and environment for statistical computing*, ISBN 3-900051-07-0, R Foundation for Statistical Computing, Vienna, Austria, 2008.

[101] S. Cass, *The 2016 top programming languages*, 2016.

[102] *The comprehensive r archive network*, http://CRAN.R-project.org/.

[103] *Mathworks file exchange*, https://www.mathworks.com/matlabcentral/fileexchange/.

[104] E. Linstead, P. Rigor, S. Bajracharya, C. Lopes, and P. F. Baldi, "Mining internet-scale software repositories," in *Advances in neural information processing systems*, 2008, pp. 929–936.

[105] E. Linstead, P. Rigor, S. Bajracharya, C. Lopes, and P. Baldi, "Mining concepts from code with probabilistic topic models," in *Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering*, ACM, 2007, pp. 461–464.

[106] T.-H. Chen, S. W. Thomas, and A. E. Hassan, "A survey on the use of topic models when mining software repositories," *Empirical Software Engineering*, vol. 21, no. 5, pp. 1843–1919, 2016.

[107] S. W. Thomas, B. Adams, A. E. Hassan, and D. Blostein, "Validating the use of topic models for software evolution," in *Source Code Analysis and Manipulation (SCAM), 2010 10th IEEE Working Conference on*, IEEE, 2010, pp. 55–64.

[108] P. F. Baldi, C. V. Lopes, E. J. Linstead, and S. K. Bajracharya, "A theory of aspects as latent topics," in *ACM Sigplan Notices*, ACM, vol. 43, 2008, pp. 543–562.

[109] A. Schofield and D. Mimno, "Comparing apples to apple: The effects of stemmers on topic models," *Transactions of the Association for Computational Linguistics*, vol. 4, pp. 287–300, 2016.

[110] *Cran - package mallet*, https://cran.r-project.org/web/packages/mallet.

[111] A. Agrawal, W. Fu, and T. Menzies, "What is wrong with topic modeling? and how to fix it using search-based software engineering," *Information and Software Technology*, vol. 98, pp. 74–88, 2018.

[112] *Zeromq package community*, http://zero.mq.

[113] *Nomad package*, https://www.gerad.ca/nomad/Project/Home.html.

[114] A. Atchison, H. Anderson, C. Berardi, N. Best, C. Firmani, R. German, and E. Linstead, "Poster: A topic analysis of the r programming language," 2018.

[115] N. Caragea, A.-C. Alexandru, A. M. Dobre, *et al.*, "R–a global sensation in data science," *Revista Română de Statistică nr*, p. 7, 2014.

[116] S. A. Culpepper and H. Aguinis, "R is for revolution: A cutting-edge, free, open source statistical package," *Organizational Research Methods*, vol. 14, no. 4, pp. 735–740, 2011.

[117] A. Decan, T. Mens, M. Claes, and P. Grosjean, "On the development and distribution of r packages: An empirical analysis of the r ecosystem," in *Proceedings of the 2015 European Conference on Software Architecture Workshops*, ACM, 2015, p. 41.

[118] R. Bååth, "The state of naming conventions in r," *The R journal*, vol. 4, no. 2, pp. 74–75, 2012.

[119] K. Hornik and F. Leisch, *Vienna and R: Love, marriage and the future*. na, 2002.

[120] K. Hornik, "Are there too many r packages?" *Austrian Journal of Statistics*, vol. 41, no. 1, pp. 59–66, 2016.

[121] *Mathworks discovery*, https://www.mathworks.com/discovery/matlab-vs-r.html.

[122] P. Martins, P. Lopes, J. P. Fernandes, J. Saraiva, and J. M. P. Cardoso, "Program and aspect metrics for matlab," in *Computational Science and Its Applications – ICCSA 2012*, B. Murgante, O. Gervasi, S. Misra, N. Nedjah, A. M. A. C. Rocha, D. Taniar, and B. O. Apduhan, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 217–233, ISBN: 978-3-642-31128-4.

[123] E. Linstead, L. Hughes, C. Lopes, and P. Baldi, "Exploring java software vocabulary: A search and mining perspective," in *Proceedings of the 2009 ICSE Workshop on Search-Driven Development-Users, Infrastructure, Tools and Evaluation*, IEEE Computer Society, 2009, pp. 29–32.

[124] W. E. Zhang, Q. Sheng, E. Abebe, M. Ali Babar, and A. Zhou, "Mining source code topics through topic model and words embedding," Dec. 2016, pp. 664–676, ISBN: 978-3-319-49585-9.

[125] M. Allamanis and C. Sutton, "Mining source code repositories at massive scale using language modeling," in *2013 10th Working Conference on Mining Software Repositories (MSR)*, 2013, pp. 207–216.

[126] F. Perez, B. E. Granger, and J. D. Hunter, "Python: An ecosystem for scientific computing," *Computing in Science Engineering*, vol. 13, no. 2, pp. 13–21, Mar. 2011.

[127] S. Ugurel, R. Krovetz, and C. L. Giles, "What's the code?: Automatic classification of source code archives," in *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '02, Edmonton, Alberta, Canada: ACM, 2002, pp. 632–638, ISBN: 1-58113-567-X.

[128] C. McMillan, M. Linares-Vasquez, D. Poshyvanyk, and M. Grechanik, "Categorizing software applications for maintenance," in *Proceedings of the 2011 27th IEEE International Conference on Software Maintenance*, ser. ICSM '11, Washington, DC, USA: IEEE Computer Society, 2011, pp. 343–352, ISBN: 978-1-4577-0663-9.

[129] D. Binkley, D. Heinz, D. Lawrie, and J. Overfelt, "Understanding lda in source code analysis," in *Proceedings of the 22nd International Conference on Program Comprehension*, ser. ICPC 2014, Hyderabad, India: Association for Computing Machinery, 2014, pp. 26–36, ISBN: 9781450328791.