Towards Robust Deep Learning Engineering: Semantic Enabled Modular Approach

by

Samiyuru Menik

(Under the Direction of Lakshmish Ramaswamy)

Abstract

Deep learning technologies have demonstrated immense capabilities in numerous domains. In the current landscape, deep learning technologies play a key role in the success of modern businesses. However, adoption of deep learning technologies has a substantial amount of untouched potential. The cost of developing custom deep learning solutions for unique business problems is a major inhibitor to far-reaching adoption of deep learning technologies. We recognize that the monolithic nature prevalent in today's deep learning applications stands in the way of efficient and cost effective customized deep learning solution development. Taking a modular approach for deep learning solution development can yield a number of advantages that ultimately helps to make deep learning solutions more accessible and widespread.

This dissertation explores the benefits and trade-offs of developing deep learning solutions taking a modular approach as opposed to taking widespread monolithic approaches while addressing key challenges of modular deep learning solution development. Towards this end, we conduct

experiments by implementing modular and monolithic solutions for representative deep learning problems and evaluate them in terms of accuracy, latency, reusability and maintainability. Our experiments show that modular solutions can be comparable to monolithic solutions in terms of accuracy while offering much desired benefits of modular solutions. However, the modular solutions often show higher latency compared to monolithic solutions. In our experiments, we show that this challenge can be mitigated by utilizing a black-box knowledge distillation approach. Furthermore, we propose a knowledge graph-based machine learning service description framework that further augment the benefits of modular machine learning solutions primarily by enhancing reusability and composability of existing machine learning modules. Finally, we propose a novel approach to develop supervised deep learning models in a modular manner enabling a number of solution engineering advantages that includes module reusability, module updates and replacement, transparency, debuggability and domain expert intervention for performance optimization while maintaining comparable accuracy and efficiency characteristics to the baseline models.

INDEX WORDS: Machine Learning, Modularity, Machine Learning Solution Engineering, Machine Learning Systems

Towards Robust Deep Learning Engineering: Semantic Enabled Modular Approach

by

Samiyuru Menik

B.Eng., University of Westminster, United Kingdom, 2015

A Dissertation Submitted to the Graduate Faculty of the

University of Georgia in Partial Fulfillment of the Requirements for the Degree.

Doctor of Philosophy

Athens, Georgia

2023

Towards Robust Deep Learning Engineering: Semantic Enabled Modular Approach

by

Samiyuru Menik

Major Professor: Lakshmish Ramaswamy

Committee: Liming Cai

Sheng Li

Electronic Version Approved:

Ron Walcott

Dean of the Graduate School

The University of Georgia

August 2023

Acknowledgments

I want to express my heartfelt gratitude for my major advisor, Dr. Lakshmish Ramaswamy, whose invaluable advice and support significantly shaped my PhD journey. My deep appreciation also goes to my committee members, Dr. Liming Cai and Dr. Sheng Li, for their valuable input and support.

Thank you!

Contents

List of Figures

Chapter 1

Introduction[1]

Machine learning (ML) has gained significant attention over the past years and ML technologies have become a part of many organizational workflows and day to day tasks of individuals. Big tech companies and academic entities are taking the lead in developing cutting edge ML technologies that push the boundaries of what ML can accomplish. Cutting edge computer vision and language modeling technologies provide a good example for this.

Beyond the heightened attention, existing applications and large scale organization and academic driven developments, there is a significant amount of untouched potential to ML. We believe that this potential lies in ground level application domains that are usually away from the mainstream attention. Consider the organizations that operate in non-tech focused business domains. These can include educational and research institutes, healthcare facilities, transportation units and government organizations. These organizations have a number of workflows that can be improved using ML technologies. Depending on the scale, these organizations are more

[1] Parts of this chapter appears in Menik and Ramaswamy, 2023

likely to have a traditional information technology and software engineering staff to fulfill the tech requirements. However, these organizations may not have the budgets, resources and expertise to focus on producing custom ML solutions for their workflows.

Making cutting edge technologies accessible to a wider range of organizations and different organizational levels and individuals that operate in a wide range of domains is one of the major challenges that ML as a field face today. Enabling wide scale adoption of ML technologies has the potential of achieving the next level of business process optimizations, service quality improvements and user experiences. As an example, today's high level decision makers of large organizations already use machine learning technologies intensively in their decision making processes. However, it raises the question that how much of these technologies are practical and cost efficient to be implemented for the use of lower levels of the organizational hierarchy. As another more concrete example, a large scale organization may implement a cutting edge machine learning solution to filter resumes in their hiring process. However, it is not practical for a much smaller scale organization to access such technologies to implement a similar system that matches their own bespoke business requirements. Recurring motif here is that, when making machine learning technologies more accessible, 1/ cutting edge technologies should be accessible to a wider audience through means that are easy to grasp 2/ it should be possible to implement customized machine learning solutions that match business requirements at a lower cost. We believe that further improving software engineering practices in machine learning solution development, especially focusing on deep learning approaches, is an effective starting point to address the aforementioned challenges.

Today's prevalent deep learning solutions are monolithic. These solutions are mostly large black boxes that produce the right answer to a given problem when fed with a large amount of relevant data. Major influence for this trend is coming from end to end deep learning technologies. Unlike traditional ML methods that involve time consuming and labor intensive feature engineering steps, end to end machine learning attempts use a larger single model that aims to learn all intermediate steps required to solve the problem at hand without needing much hand engineering during the learning process. This is usually achieved by employing large parameterized models that can take advantage of large amounts of training data. This way of developing ML solutions significantly cut down the human effort needed to implement ML solutions that produce impressive results for a variety of ML problems. However, this approach of developing ML solutions has a set of key disadvantages as well. They became especially apparent when developing ML solutions outside of cutting edge tech focused organizations.

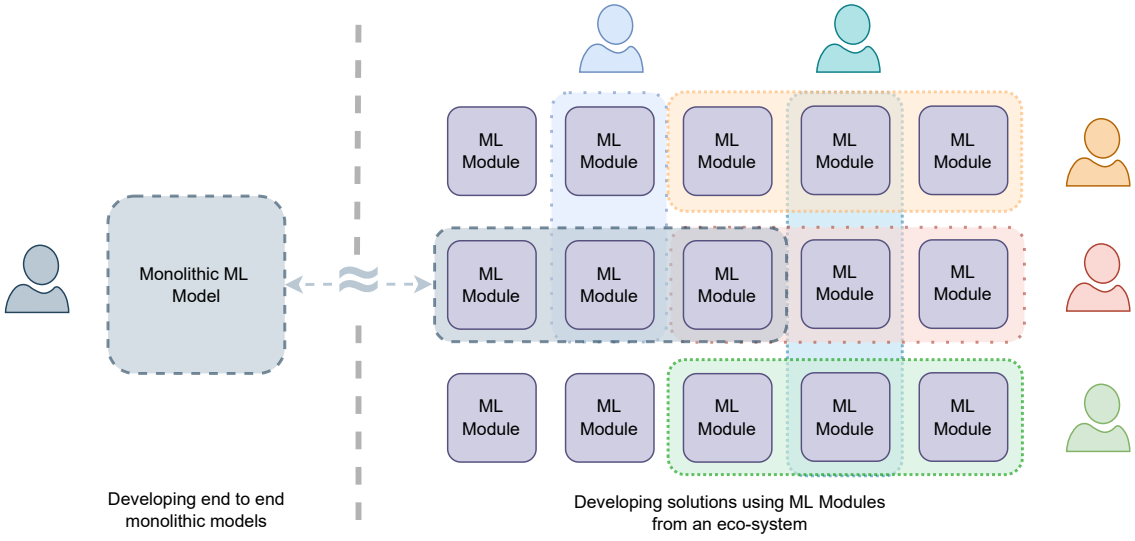

Figure 1.1: Visualization of monolithic ML solution development in comparison to modular ML solution development in an echo-system. In the modular solution development paradigm, ML modules are reused to develop different solutions by different users enabling enhanced accessibility to technologies.

One of the key limitations of prevalent end to end machine learning solutions is the lack of modularity. This results in a number of other critical engineering challenges. Usually one monolithic solution is only applicable to one specific problem. With that combined with the inability to break down the solution into semantically meaningful and more generally applicable submodules, the effort that went into a monolithic system can not be easily reused in other contexts. Also, it is not possible to replace parts of the system as new technologies are introduced with improved performance characteristics. Further, the solutions usually have to rely on datasets that are specifically focused on the problem at hand.

Traditionally engineering practices encourage modular solutions to overcome the aforementioned challenges and develop more maintainable and cost efficient solutions. We believe that encouraging modularity in deep learning solutions can bring a number of advantages to deep learning solution development that other more traditional engineering domains already experience. Fig. 1.1 shows a visual representation of monolithic ML solution development in comparison to modular ML solution development.

As an initial step towards promoting software engineering practices by focusing on modularity, we explore the cost benefit trade-offs of solving machine learning problems in a multi stage modular way in comparison to solving them with a conventional monolithic deep learning solution. We perform this analysis by implementing deep learning solutions in two ways for three representative problems. For each problem, one solution is developed taking a monolithic approach. The other solution is developed taking a modular approach. Then, we evaluate the two approaches quantitatively and qualitatively to analyze the trade-offs of each approach and address key challenges. Next we focus on addressing the challenge of publishing, discovering, reusing and

composing machine learning modules by introducing a framework for describing machine learning modules using knowledge graphs that can be used as a key component of machine learning as a service paradigm. Finally, we propose a new approach for enabling modularity in supervised deep learning solutions that enable benefits of modularity that we explored throughout this work and experimentally study the capabilities of the proposed approach.

1.1 Challenges of Modular Deep Learning

Modularizing deep learning models offers significant benefits in terms of solution engineering. However, there are several key challenges associated with modular deep learning. In a modular deep learning system, the modules play a key role in the solution. Because of that it is crucial for a modular deep learning system to have a rich ecosystem of modules as depicted in fig. 1.1. In such an ecosystem, modular deep learning systems should have a way of module discovery, reuse and composition. These capabilities depend on having a robust module description framework that is capable of specifying important aspects of deep learning modules. Further, modular deep learning solutions should have comparable performance characteristics to the conventional monolithic deep learning solutions while delivering the benefits of modularity. This is challenging since modularity is another constraint on the solution design that limits solution design space. The effect of this limitation should be addressed by studying the trade-offs between modular and monolithic deep learning solutions and offering solutions to mitigate potential downsides of modular deep learning. In addition to that, transparency and maintainability are two major expectations of modular deep learning solutions. In order to facilitate this, it is important for modular deep learning

solutions to incorporate high level semantic meaning to the modular substructure of the solution. With that, the modular deep learning solutions should provision enhanced troubleshooting and module maintenance at the level of individual modules while ensuring the overall performance characteristics of the solutions.

1.2 Research Objectives and Contributions

The principal objective of this dissertation is to explore the capacity of modularity to address solution engineering challenges that arise in the development of deep learning solutions. Further, this dissertation aims to address the primary challenges that were previously mentioned regarding employing modularity in deep learning solution development. Towards this end this dissertation presents the following key contributions.

- Highlighting the need for modularity in deep learning by drawing attention to the advantages of modularity in deep learning solution development compared to conventional deep learning approaches.
- Highlighting the broader impact of modular deep learning solutions by emphasizing the role of modularity for the accessibility and wide scale adoption of the technology.
- Conducting empirical studies to analyze the trade-offs of modular deep learning solution development compared to conventional monolithic deep learning solution development.

- Identifying high-level semantic meaning, module discovery, composition and performance overhead as key challenges of modular deep learning through experimental analysis using case studies.
- Proposing a framework to describe machine learning modules using knowledge graphs to improve publishing, discovery, reusability and composition enabling high level semantics for existing machine learning modules.
- Proposing a novel approach to develop modular deep learning solutions for supervised machine learning problems using soft-switches to enable high level semantic meaning for the components used in the deep learning model, enable reusability and composition and enhance transparency and maintainability.
- Conducting empirical studies on the proposed modular deep learning model development approach with benchmark datasets and analyzing the effectiveness of the approach.

1.3 Dissertation Organization

The rest of the dissertation is divided into the following chapters. Chapter 2 presents relevant background material and related work. Chapter 3 focuses on benefits and trade-offs of modular deep learning solutions. In this chapter we identify the major latency drawback of modular solutions and address them using black box knowledge distillation. In chapter 4 we propose a knowledge graph based machine learning solutions description framework that enhances publishing, discovery, reuse and composition of machine learning services as a main component of machine learning as a service paradigm. Chapter 5 presents a modular deep learning architecture

that uses a soft switch to enable semantic modularity in supervised machine learning problems. The dissertation concludes in chapter 6 with a summary of the dissertation and directions for future work.

Chapter 2

Background and Related Work[1]

This chapter aims to set the context for modular deep learning solution engineering by providing background information. It explores the concepts of modular deep learning and deep learning model development while focusing on reusability and composability of deep learning modules. Additionally, the chapter provides an overview of existing work in this field. This chapter provides a foundation for the forthcoming chapters that discusses the proposed approaches on modular deep learning solution engineering.

2.1 Software Systems Engineering

The concept of modularity takes center stage in software engineering and it spans from conventional software solutions that sit on a local machine to distributed web services and relatively newer evolutions of them such as microservices architectures. Early work such as Parnas, 1979 promoted the discussions of modularity in software programs and introduced practices that en-

[1]Parts of this chapter appears in Menik and Ramaswamy, 2021, 2023

hance software engineering in general. Beyond such literature, a large portion of engineering practices around modularity in software engineering come from trial and error driven hands-on experience developing systems in the industry and academia over the years. In the context of web services, service oriented architectures (SOA) Papazoglou and Van Den Heuvel, 2007 have emerged as a form of high level modularity. In this paradigm, large scale software systems are organized into services where individual services can be independently developed, deployed and consumed. This paradigm promotes reusability and composition at an industry scale. The microservices architectures Jamshidi et al., 2018 can be seen as a more fine grained version of service oriented architectures. Microservices architecture promotes further breaking down large software systems into smaller more manageable components that can be independently developed, deployed and scaled as needed. With the success and large-scale adoption of these paradigms, a number of other major concerns such as interoperability, service discovery, composition definition and deployment concerns and effective solutions for them have emerged over time. The concerns and solutions around these paradigms are conceptually valid for machine learning solutions at a high level. However, the concrete ways to effectively apply them and the effectiveness of such applications in the context of machine learning, especially deep learning, needs further research.

2.2 Modular Deep Learning

Current prominent deep learning approaches are monolithic. They are black boxes with no semantically meaningful substructure. The substructure that these models have are too low level to associate any high level meaning. Consider widely adopted computer vision deep learning ar-

chitectures such as ResNet He et al., 2016, DenseNet Huang et al., 2017 or U-Net Ronneberger et al., 2015. These models are composed of low level primitives such as convolutions, pooling units, linear units, activation functions etc. Beyond that these architectures also have repeating structures that run from input to the output. However, it is difficult to point out to different parts of the model and point out what high level function that part is responsible for. Same applies to recent transformer architectures Devlin et al., 2018; Touvron et al., 2021; Vaswani et al., 2017; Yang et al., 2019. Regardless of this general lack of modularity there are several lines of existing work that introduces modularity and substructure to deep learning models.

There have been attempts to break pre-trained neural networks into a set of modules based on the learned weights. Ultimate goal of this is to find semantic modules within a learned neural network. Csordas et. al. Csordás et al., 2021 have attempted to do this by learning weight masks to identify subnetworks for target tasks. They were able to find some specialized subnetworks in trained neural networks with some other interesting insights. However, to the best of our knowledge, these works have so far not been able to find strong semantically meaningful modules within learned monolithic neural networks that can be reused in other contexts.

Ensemble learning methods Sagi and Rokach, 2018 has a sense of modularity. An ensemble model is not a single monolithic unit. Ensemble methods like bagging, stacking and boosting combine several machine learning models to create a more robust single solution. In these methods each model in the ensemble will be specialized in one aspect of the whole problem. These specialized parts together create a better final solution to the overall machine learning problem. However, the modules in ensemble models are usually not reusable in other systems since they are specialized to a given specific ensemble that solves one problem. Another line of work that

attempts to incorporate modularity in deep learning is routing networks Cases et al., 2019. Modular question answering approach proposed by Jacobs Andreas et al., 2016 is another example for similar work. This line of work mainly tries to learn a set of modules and a controller to compose these modules to solve problems. This learning process is usually done in an end to end fashion. Even though these approaches have shown good performance in the problems that the model was trained to, the modules have not shown to be much useful outside of the problem in concern. Therefore, routing networks, in their current state, are not yet capable of addressing the problems that we discussed earlier.

Transfer learning techniques Zhuang et al., 2020 allow machine learning model developers to train customized solutions with less amount of training data by fine tuning an already existing machine learning model that is usually pre-trained with a large dataset. Transfer learning has shown to be very effective when an already existing model is similar to the target task. There are few downsides to transfer learning approaches. First, the model developers usually have to be knowledgeable about the pre-existing model internals in order to be able to modify it to match the target machine learning problem. The modified model has to be retrained with new training data that better represent the new machine learning problem. This is different from traditional software engineering modules that attempt to expose a clean interface to the users by hiding the module internals. When it comes to making machine learning technologies more accessible to a wider audience, transfer learning techniques should be further improved to enable a more ready to use modularity that minimizes the chances of developers having to work with the internals of complex network architecture.

2.3 Deep Learning Model Engineering

The most fundamental form of developing deep learning models is to start with primitive deep learning operators and develop an architecture from the scratch to process the desired input and produce a representation of the desired output. Then the model has to be trained with an appropriate dataset with a suitable loss function and an optimization algorithm. However, this way of developing a deep learning model is very complex and costly in terms of time and other resources since there is a very large number of decisions to be made to develop a successful model this way. These decisions range from selecting the right primitives and combining them in the right way to selecting suitable loss functions and optimization algorithms to selecting proper hyper parameters. Some of this burden can be eased by using one of many APIs and libraries like Keras Chollet et al., 2015 and Mllib Meng et al., 2016 for developing deep learning models with implementations for a large number of primitive operators. These APIs and libraries provide high level abstractions for primitives and provide various integration capabilities. However, these libraries do not address the concern of semantically meaningful substructure of resulting models. Beyond this there are several other prominent resources and approaches that can be used to develop deep learning models.

ML model and dataset repositories such as OpenML Vanschoren et al., 2013 and Google AI Hub "AI Hub | Google Cloud", 2020 and related search engines attempt to promote reusing existing ML models and datasets. However, many of these solutions are limited by simple keyword based search mechanisms and lack of descriptions about the models and datasets.

Cloud based ML platforms like Google AI Platform "AI Platform | Google Cloud", 2020 eliminates a considerable amount of systems related concerns from the users. But these solutions have not adequately eliminated ML related complexities from the users. On the other hand, ready to use web based ML services are not sufficiently flexible to be used in many real world use cases.

ML model specification approaches such as ONNX Bai et al., 2019 and ML-Schema Publio et al., 2018 focuses on promoting reusability and composability of ML models by describing various aspects of ML models. However, these solutions are overly complex or do not cover all major aspects that are discussed in this paper. Several works in model selection and meta learning Nural et al., 2017; Vanschoren, 2018 also attempt to describe ML models. But these solutions are more focused on the model selection aspects rather than model description. Another limitation in some of the previous work is that they are more focused on the low level features of the models and data sets Vanschoren, 2018. They do not emphasize high level semantic information regarding the models that are important for domain experts. Further, some of these works are dated and they have not accounted for the advances in deep learning.

Several industrial solutions have used simple and less formal machine learning module description methods. Google AI Hub "AI Hub | Google Cloud", 2021 and similar industrial solutions use simple sets of key values to describe modules. These descriptions expose some level of module semantics. Commonly used keys include category that describes the type of the module ie: TensorFlow module etc., publisher, input data type, machine learning techniques that have values to describe the machine learning techniques used in the module ie: convolutional neural networks, transfer learning, embeddings, etc. This key value information is used for searching and information providing purposes. In addition to the key value information they also provide a textual

description of the important information such as an overview and a brief guidance on how to use the module. The advantage of using simple key values is that they are easy to be added to machine learning modules at scale. Further, the systems that are required to process the key values as an example for searching are also less complex.

Authors of AIMMX Tsay et al., 2020 argue that development of artificial intelligence solutions is currently a fairly ad-hoc process even though most of the developments are done using general software engineering tools. The authors point out that the lack of information about machine learning components is hindering getting insights into the state of the art artificial intelligence developments in order to solve the aforementioned issue. With this argument, Tsay et al. introduces AIMMX Tsay et al., 2020 a tool that can be used to automatically extract metadata from machine learning software repositories. As a part of this solution authors have extracted a large number of machine learning related software repositories through sources like Model Zoos, arXive papers and state of the art papers. Then, the software repositories are processed to extract model meta data such as the name, references, associated datasets, machine learning frameworks, the associated domain of the machine learning model and other information. Later the authors have produced human readable and machine processable metadata for the machine learning models in the repositories. Further, the authors have shown some of the use cases of the extracted metadata. Metadata that was extracted by AIMMX contains semantic information about models. In addition to that since the extraction is automatic the process can be performed at scale to cover a large number of machine learning components. However, the metadata extracted by AIMMX is not sufficient to improve discoverability and reusability of existing machine learning models.

When it comes to complex machine learning models used in difficult problem domains, manual architecture design and hyper-parameter tuning is becoming increasingly infeasible. Recent developments in automated machine learning (AutoML) attempts to solve this challenge by automating the development of ML models by automatically generating neural architectures and tuning hyper-parameters with the objective of achieving optimal performance for a given problem. AutoML addresses neural architecture search and hyper parameter tuning problems by taking a systematic search approach. AutoML uses approaches like grid search Bergstra and Bengio, 2012; Komer et al., 2014; Lin and Zhang, 2013, randomized search L. Li and Talwalkar, 2020; L. Li et al., 2017, bayesian optimization, gradient descent methods Liu et al., 2018; Maclaurin et al., 2015, reinforcement learning methods Wu et al., 2016; Zoph and Le, 2017, evolutionary methods Stanley and Miikkulainen, 2002; Suganuma et al., 2018; Xie and Yuille, 2017 to perform the search and optimize the model architecture and the hyper parameters. These autoML approaches have shown to produce very successful outcomes over the years. However, these techniques are often very resource intensive and costly to be employed in domain applications. Further these solutions do not provide a clear path towards deep learning models with semantically meaningful substructure.

2.3.1 Machine Learning Component Discovery

Discoverability is a key enabler of reuse of existing components when developing deep learning models. Web service discovery is a close relative to machine learning service discovery. In the web services domain, services are described, discovered and accessed using technologies and associated open standards like XML, WSDL, UDDI, SOAP, REST, and HTTP. These technologies and

standards are widely used in real world settings. There are many instances that machine learning services are implemented and served behind aforementioned web service infrastructures. Azure cognitive services Machiraju and Modi, 2018 is a widely used industrial example for a set of ready to use machine learning inference services behind a web service infrastructure. These types of machine learning services are easy to use and harness the benefits of traditional web services including discoverability. These services can be published in web service registries as traditional web services and can be discovered by the consumers. However, one major disadvantage of these services is that there are only a limited number of such services available for a set of common machine learning problems such as basic language, vision and speech tasks. It is not realistic to expect service providers to have services for all possible requirements of different users. Therefore, these services fail to effectively serve custom machine learning solution needs of domain experts.

Another popular method that is ready to use machine learning solutions are available in repositories with search capabilities. OpenML Vanschoren et al., 2013 is an open platform that enables sharing machine learning models, pipelines and datasets. Open ML contains a number of machine learning models associated with datasets and their performance results. This makes it possible for practitioners to compare models and reuse the ones that fit the needs. Open ML has integrations for popular machine learning frameworks to make sharing and reuse seamless. Google AI Hub "AI Hub | Google Cloud", 2021 is a commercial repository that contains machine learning models that are trained and untrained. These machine learning models can be used by loading the models into projects as they are or by retraining. ModelZoo "Model Zoo - Deep learning code and pretrained models for transfer learning, educational purposes, and more", 2020, ModelDB Vartak et al., 2016 and DLHub Chard et al., 2019 are similar machine learning model reposito-

ries that provide a central hub to discover existing machine learning models for various projects. Most of these repositories do not have comprehensive descriptions of machine learning models making reusing difficult. Further, since these models are available as raw modules that are specific to certain machine learning frameworks there are several prerequisites including the expertise in the required technology that limits reusability. These concerns add additional constraints to the discovery process limiting the usability that are not existent in traditional web service discovery. Most of the machine learning model repositories have simple textual search mechanisms. These do not support performing effective advanced searches to locate specific machine learning models for use cases in different domains.

Most of the popular machine learning frameworks are bundled with a set of commonly used machine learning models and datasets. As an example Keras Chollet et al., 2015 provides a set of popular machine learning models readily usable within the framework. It has a number of models such as NASNetLarge and InceptionResNetV2 with and without training parameters. With the models corresponding datasets are also often available. These frameworks enable a type of model discoverability in machine learning projects. Some machine learning frameworks have automatic model selection capabilities. As an example the Scikit-Learn Pedregosa et al., 2011 provides tools that support model selection such as cross validation and various configuration search methods. These tools can be utilized along with available existing models to find the best performing model for the problem at hand. Even though these solutions are very effective in practice, they need technical expertise in machine learning that many domain experts do not have. Further, the number of existing models that are available in machine learning frameworks are very limited.

There are attempts of crawling existing machine learning component repositories and extracting meta information out of the components to create catalogs of machine learning components to enable discoverability and reuse. These tools have the capability of scanning multiple sources from different component providers. Further, these solutions have the capacity of indexing a large number of machine learning models at scale. AIMMX Tsay et al., 2020 is an implementation of this approach. AIMMX has a web application view similar to Google AI hub "AI Hub | Google Cloud", 2021 to search and explore existing machine learning models crawled by the tool. One limitation of these tools lie in the metadata extractors. There is a number of important metadata information that is hard to extract from component repositories without explicit information. The lack of information in extracted models limits the effectiveness of the approach in discovery and reusability of existing machine learning models.

Chapter 3

Modular Machine Learning Solution Development: Benefits and Trade-offs[1]

3.1 Modularity in Machine Learning

Modularity is a familiar approach when solving complex problems in many domains. It simplifies the problem solving process by breaking down a complex problem into more manageable subparts. After breaking the problem into subproblems the subproblems can be solved independently. This process brings a number of advantages to the solution engineering process. The goal of making ML solutions modular is to enable these advantages to ML solution engineering.

[1]Major parts of this chapter appears in Menik and Ramaswamy, 2023

Combinatorial generalization is one of the main advantages of modular ML models. In general an ML model that is trained for a specific problem is only useful for that specific problem. Therefore the monolithic models that can not be broken down into submodules are not usable outside of the problem that it intends to solve. On the other hand modular ML models are composed of more than one ML model that each model solves a sub problem of the original problem. This allows parts of the modular ML models to be reused in different contexts beyond the original problem. This enables mixing and matching modules to create new unique models to solve different problems. This helps to minimize the chances of having to start from the scratch when developing unique solutions. On the other hand, when using end to end learning methods that learn a monolithic model, very often, each new problem is a unique problem that has to be solved starting from fundamental technologies.

The same is true for datasets that are used to train ML models. Training monolithic models require a dataset specific for the intended problem that each datapoint maps a specific input type to a specific target type with other required characteristics. Such a dataset is only useful to train a model similar to the original problem. In comparison, modular ML models are a composition of submodules where each model is trained with a dataset that matches the subproblem. At the same time, the process of breaking down a larger problem into subproblems usually results in subproblems that are simpler and more general. Therefore, within a development ecosystem, modular ML solutions increase the chances of reusing datasets to solve different problems.

Domain expert intervention to simplify the learning process by incorporating domain knowledge when developing solutions. One of the ways to do this is by decomposing the problem in a way that the resulting subproblems are more data efficient and simpler to learn. As a simple

example, think about a multi-digit classification problem. In this case, a domain expert may break down the problem to detect each digit individually using a simpler model and later aggregate the classification results to find the classification for the original multi digit input. This simple decomposition made the subproblem simpler as well as more data efficient while increasing the amount of data points per classification class. In traditional software engineering modularity allows engineering teams to divide work across individuals or specialized teams. This minimizes the development overhead by assigning units of work with cross-cutting concerns to one individual or one team. In this process a module boundary can define a unit of work that can be assigned to an individual or a team. In addition to that this helps to reduce dependencies among units of work and parallelize the development process.

Performance tuning ability is higher with modular ML models. Modularity enables breaking down a problem into subproblems and addressing them separately. Since subproblems tend to be more general than the overall high level problem, finding technologies and existing solutions for the subproblems is likely to be easier. This enables more options for solution developers to make performance trade-offs at the subproblem level. Further, modularity makes it easier to do incremental improvements to solutions. Individual modules in modular systems can be replaced with different modules with the same functionality but with different characteristics. As an example, one may trade off accuracy for faster performance at one subproblem of the modular solution to meet a business requirement at hand. With monolithic solutions, making performance trade-offs usually require replacing the whole model with a one that has the required characteristics.

Maintainability of modular ML models is higher compared to monolithic ML models. Since the submodules of modular ML models can be replaced with different models with the same

functionality, newer or improved technologies emerge, submodules can be upgraded without requiring major changes to the entire solution. In addition to that, since modular models are a composition of semantically meaningful models, they are more human understandable. This opens up more opportunities to verify and monitor modular models. This simplifies the process of isolating issues and troubleshooting.

Economies of scale effect can be harnessed better with modular ML models from a development ecosystem perspective. Since modular ML models can take better advantage of existing ML modules by reusing them to create different higher level solutions, the modules that are more commonly reused in many problems get a higher demand from the development community. As a result of this, more demanding modules are likely to be further improved within the ecosystem due to the economies of scale effect and these improvements can be exploited by the downstream solutions. On the other hand, in end to end ML, this effect is relatively less prominent since the learned solutions are monolithic and specialized to individual problems making them less usable in other contexts.

3.2 Case Studies

As discussed before, we will be using three example problems in order to empirically highlight the benefits and trade-offs of modular and monolithic machine learning solutions. First example problem is a text based sentiment analysis. The other two problems are a satellite image classification problem and a near infrared (NIR) field prediction problem.

3.2.1 Text based Sentiment Analysis

Text based sentiment analysis is useful in a number of business contexts. For instance, businesses use sentiment analysis to understand consumer sentiment towards their brand and the products. In today's internet based global market setting, analyzing the sentiment of a text in a given language is important for many business organizations. In this section we are using this problem as a proxy to study the trade-offs of monolithic and modular machine learning solutions.

Deep learning technologies have demonstrated state of the art performance in sentiment analysis tasks. Current prevalent deep learning based models are monolithic in nature Devlin et al., 2019; Tan and Le, 2019. Monolithic deep learning solutions for sentiment analysis train machine learning models end to end to predict the sentiment of a text from a given source language. More modular solution for sentiment analysis is to develop the solution using two modules that solve the problem in two intuitive stages. The first stage is to translate the source language text to a suitable target language. The second stage is to analyze the sentiment of the translated text. This approach enables the opportunity to train a sentiment analysis model for a language with a larger and more representative sentiment analysis dataset or to find a sentiment analysis model already trained for a specific language that demonstrates higher performance characteristics. In this section we will be implementing two sentiment analysis solutions one monolithic and one modular as described before and compare the advantages and disadvantages of the two approaches.

This experiment is performed with a Spanish sentiment analysis problem. Off the shelf pretrained language models are used to implement the monolithic solution and the two stage modular solution for this problem. The monolithic version of the experiment is conducted using an exist-

ing BERT based pretrained multilingual sentiment analysis model Wolf et al., 2019. The model is already trained with 150k English, 80k Dutch, 137k German, 140k French, 72k Italian and 50k Spanish sentiment analysis data points. The model predicts the sentiment of the input text in a 1 to 5 scale where 1 is the least positive and 5 is the most positive sentiment. The first stage of the modular version is implemented with an existing pretrained spanish to english translation model Tiedemann, 2020; Wolf et al., 2019. The second stage uses the same sentiment analysis model that is used in the monolithic version. Further, the monolithic solution and the modular solution were distilled Hinton et al., 2015 into a smaller model to see the performance characteristics of the distilled solutions in comparison. At a high level, the distillation process is analogous to program compilation in software development. In software development, the result of compilation is an object that runs much faster during deployment. However, the compiled object is less meaningful to humans compared to the program written using a high level programming language. Similarly, in the case of a modular ML solution, the distillation results in a faster solution but compromises the explainability that is present in the modular solution. Distillation is performed with a smaller convolutional architecture as the student network in both monolithic and modular cases. Input to this model is an integer sequence that was created using a word dictionary that contains a unique index for each word in the corpus. Inputs to the student models are truncated to a maximum length of 500 words and padded appropriately if a sequence is short. The architecture of the distillation student network is shown in fig. 3.1. The distillation is done by considering the teacher model as a blac kbox to keep the process more generally applicable. In the distillation process the student is trained to imitate the teacher by minimizing the cross entropy loss between the teacher

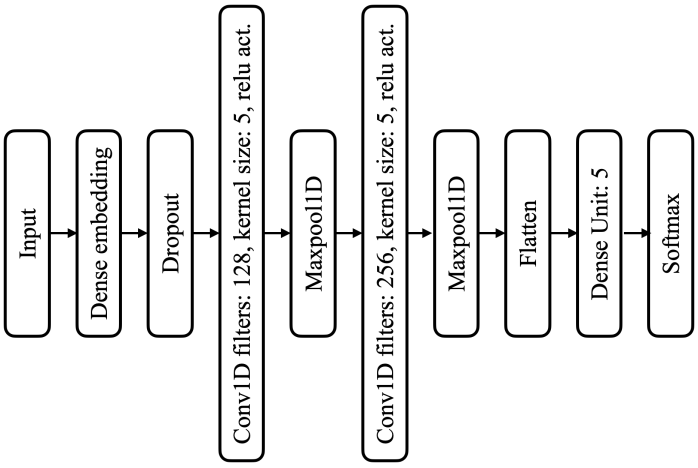

Figure 3.1: Architecture of the sentiment analysis student model.

model's output and the student model's output. This distillation process does not require any labeled data points. It only needs unlabeled text from the input language.

Fig. 3.2 shows a diagram of the monolithic and the modular solutions used in the experiment with their distilled counterparts.

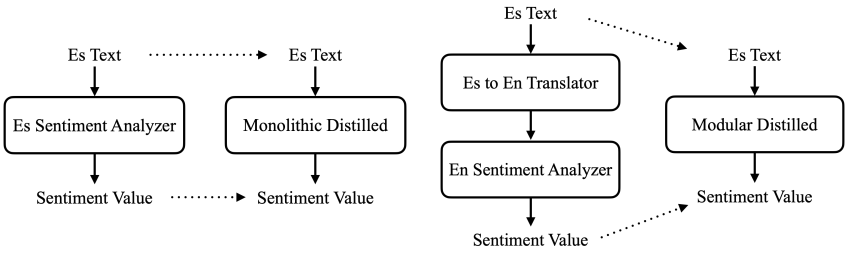

Figure 3.2: Monolithic (left) vs modular (right) sentiment analysis models and the models distilled from them.

Performance characteristics of these models are compared using the test set of the spanish portion of the amazon multilingual product review sentiment dataset Keung et al., 2020. This contains 30000 sentiment analysis data points. Each datapoint has spanish product review text and a 1 to 5 star rating that correspond to the product review text.

3.2.2 Satellite Image Classification and NIR prediction

Satellite image based remote sensing is useful in a number of real world applications such as land survey, surveillance, traffic monitoring etc. In this section we are studying the trade-offs between modular vs monolithic ML solutions using a satellite image classification problem. In this problem we are classifying cloudy satellite images based on the EuroSAT dataset Helber et al., 2019 into 10 classes of land use and land cover. We will be implementing one monolithic model and two modular models for this classification problem. Performance of these three models will then be evaluated based on accuracy and latency. The accuracy of the solutions are also compared with a large percentage of weight pruning. Further, we evaluate the performance of models distilled from the monolithic and two modular models. Next, we reuse a module from the modular classification solutions to predict the near infrared (NIR) band of the EuroSAT based cloudy satellite images. This modular NIR band prediction model is compared with a monolithic model for the same task. Unlike in the previous example, in this example we are training custom ML models for the problem.

EuroSAT dataset has a red green blue (RGB) version and a 13 band version. For our experiments, we use the RGB and NIR bands in the EuroSAT dataset. EuroSAT dataset only contains clear satellite images without obstructions. We alter this dataset by adding a cloud overlay to the RGB bands using the approach proposed by Kenji et al. Enomoto et al., 2017. Fig. 3.3 shows a sample of the altered EuroSAT dataset. Added clouds makes the dataset more challenging for prediction tasks compared to the original EuroSAT dataset. Since we want to represent the low labeled data regime that is commonly seen in real world industrial application domains, we use

Figure 3.3: EuroSAT data points before (top row) and after (bottom row) adding cloud layers.

only a 20% of the EuroSAT labeled data to train, validate and test the models for the classification and NIR band prediction tasks. In the case of the classification task we use the cloudy RGB image as the input and the corresponding classification label as the target. In the case of NIR prediction, the cloudy RGB image is used as the input and the corresponding NIR band is used as the target. The rest of the data that is not used for the classification and NIR prediction is used to train, validate and test the cloud removal module that is used in the modular models. It should be noted that this training step does not utilize the labels from the original EuroSAT dataset. This cloud removal dataset has the RGB images with the cloud overlay as the input and the corresponding cloud free RGB images as the target. Such an unlabeled dataset is relatively easy to acquire in larger quantities in practice in the real world as well since it does not involve manual data labeling.

Our monolithic classification model is trained end to end, validated and tested using the classification split of the cloudy satellite images. The network architecture used in the monolithic model is shown in fig. 3.4. This architecture downsamples the feature maps while increasing the number of channels as the layers progress from input to the output. In the final layers the

output of the convolutional layers are flattened and sent through dense layers to do the classification. Dropouts are used before the features are fed to the dense units. The modular versions

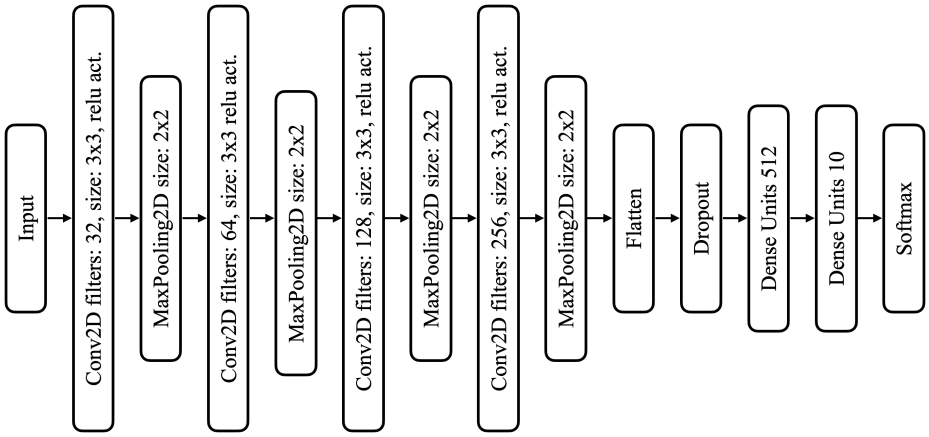

Figure 3.4: Architecture of the satellite image classification model.

perform the classification in two stages. The first stage performs cloud removal. Cloud removal is performed using the encoder decoder network architecture that is shown in fig. 3.5. First part of this architecture, the encoder part, downsamples the feature maps using 6 downsampling blocks. Each downsampling block has a convolutional layer, batch normalization layer and a leaky relu activation. The first downsampling block does not have batch normalization. Each downsampling block from input to output reduces the feature map size while increasing the number of channels. The second part of the architecture, the decoder part, upsamples the feature maps that are downsampled by the decoder using four upsampling blocks. Each upsampling block has a convolutional transpose layer, dropout layer and a relu activation layer. The first upsampling block does not have a dropout layer. Each upsampling layer from input to output increases the width and the height of feature maps while reducing the number of channels. As shown in the architecture figure, skip connections are used from the downsampling blocks to upsampling blocks with a mirror-like correspondence to incorporate low level features to the latter upsampling layers.

Finally the output of the upsampling layers are sent through another convolutional transpose layer with three channels and a sigmoid activation function. This architecture is adapted from the U-net Ronneberger et al., 2015 and pix2pix Isola et al., 2017 architectures. The second stage

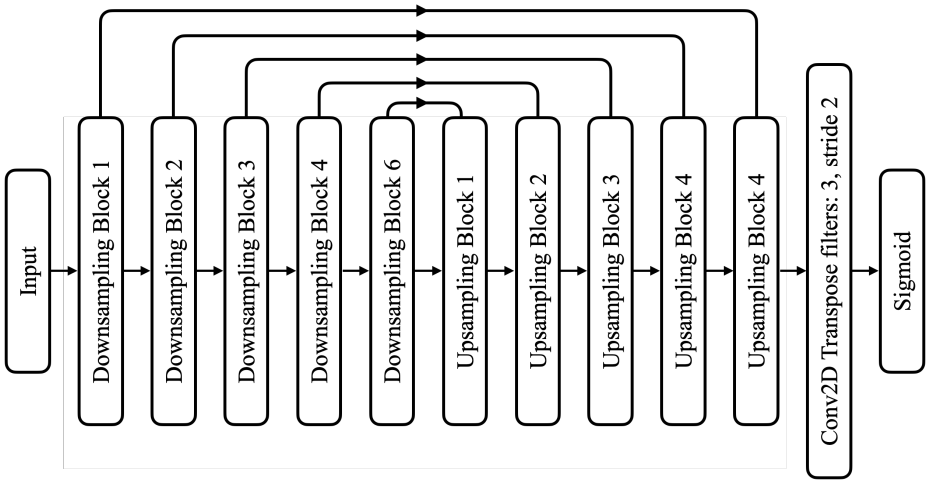

Figure 3.5: Encoder decoder architecture of the cloud removal model.

performs classification on the cloud removed images. In the first modular solution the classifier is trained using the output of the cloud removal module. This modular version represents the case where the classification data available for training is cloudy. We will be calling this solution modular (I). In the second modular version the classifier is trained using cloud free satellite images. This modular version represents the case where the classification data available for training are cloud free. We will be calling this solution modular (II). This corresponds to making a barebone pretrained satellite image classification module to be published in a model repository to be used by others. All classifiers have the same network architecture shown in fig. 3.4 and they are trained using the classification data split that was discussed before. The cloud removal module is trained with the remaining data split that was described before. The output of the cloud removal module is shown in fig. 3.6. Further, three distilled models are trained from each monolithic, modular (I)

Figure 3.6: Output (bottom row) of the cloud removal module when it is fed with cloudy data (top row).

and modular (II) solutions. Distillation is done by using the same approach that was used in the sentiment analysis case. However, before the distillation, student models that correspond to the monolithic solution and the modular (I) version are pre-trained with the available labeled cloudy classification data. Student model of the modular (II) version is not pretrained this way because it represents the case where cloudy labeled classification data is not available.

3.3 Trade-offs of Modularity

In this section we will use the solutions we developed for the three example problems to compare and contrast the trade-offs between monolithic and modular solutions.

3.3.1 Accuracy

This section compares the accuracy of the monolithic and modular solutions using solutions developed for the sentiment analysis and satellite image classification problems.

Sentiment Analysis

For the sentiment analysis case, the spanish product review test set from the amazon multilingual product review sentiment dataset is used to compare the accuracy of the two original monolithic and modular solutions and the two distilled versions of the solution. In this experiment we use a one-off accuracy measure to quantify the performance. Here we consider a prediction of the model as correct if the predicted star rating is exact or off by only one. If not, we consider the prediction as incorrect. We believe that this measure is more realistic because there is no universally agreeable star rating for a given review text. The results of this experiment are shown in fig. 3.7.

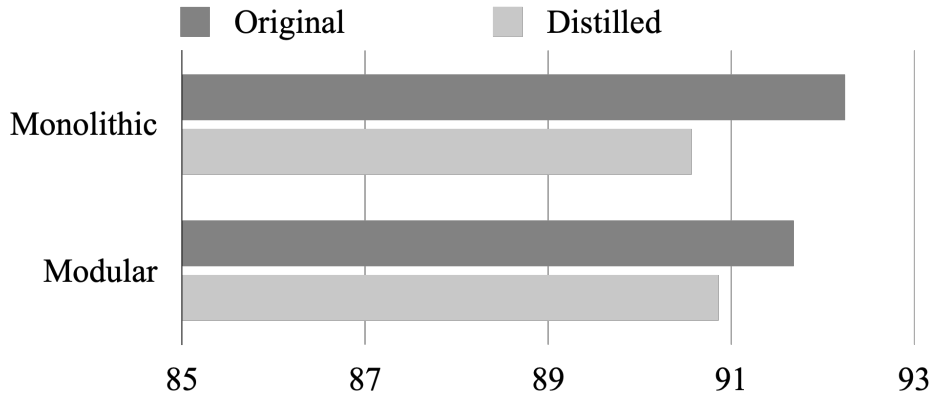

Figure 3.7: Sentiment analysis accuracy of monolithic and modular solutions including their distilled versions.

The results in fig. 3.7 shows that the original monolithic solution has less than 1 percentage point higher accuracy over the original modular solution. The solution distilled from the

monolithic solution is less than 2 percentage points lower in accuracy compared to the original monolithic solution. The model distilled from the modular solution is less than 1 percentage point lower compared to the original modular solution. The model distilled from the modular version has 0.3% higher accuracy compared to the model distilled from the monolithic solution. The results show that modular solutions can be comparable in terms of accuracy to monolithic solutions.

Satellite Image Classification

The test split of the EuroSAT dataset is used to measure the accuracy of the models and the results are shown in fig. 3.8.

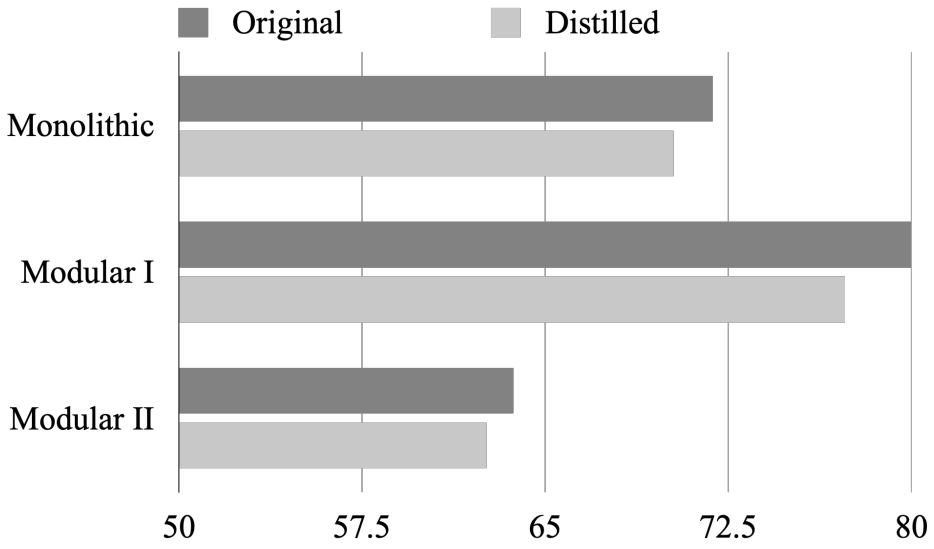

Figure 3.8: Cloudy satellite image classification accuracy of monolithic and modular solutions including their distilled versions.

The results in fig. 3.8 shows that the modular (I) solution that was trained with cloudy labeled data with the added cloud removal module performs the best out of the three solutions. It has been able to achieve 11.34% accuracy improvement over the monolithic solution when comparing the

original non-distilled solutions. When comparing the distilled solutions, the modular (I) solution was able to achieve 10% accuracy improvement over the corresponding monolithic solution. This accuracy improvement can be attributed to the cloud removal module in the modular version that was capable of utilizing low cost unlabeled data. Modular (II) solution in which the classification module is trained with clean data has the lowest accuracy. However, this solution attains its accuracy level without using any labeled data points from its target problem, cloudy satellite image classification.

3.3.2 Latency

This section compares the latency of the monolithic and modular solutions using solutions developed for the sentiment analysis and satellite image classification problems.

Sentiment Analysis

The spanish product review test set of the amazon review dataset is used to compare the latency of the two original monolithic and modular solutions and the two distilled versions of the solution. In this experiment we measure the time it takes for each model to process the 30000 test data points. The results are shown in fig. 3.9. The experiments are conducted on a machine with Intel(R) Xeon(R) CPU @ 2.20GHz, 13298580 kB of RAM and a Tesla P100 16280MiB GPU.

The results in fig. 3.9 shows that the original modular solution is much slower compared to the original monolithic solution. This performance drop is mainly due to the translation model used in the first stage of the modular solution. However, the solutions distilled from each of the original solutions are much faster as we can expect. These results suggest that developing modular solutions

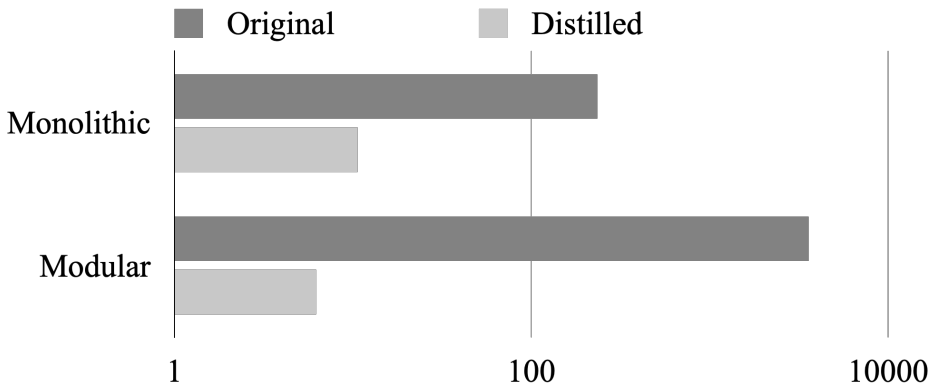

Figure 3.9: Sentiment analysis latency of monolithic and modular solutions including their distilled versions.

and distilling them into smaller models can result in efficient solutions with minor accuracy trade-offs. The additional benefit of this approach is that the modular solution development provides a number of engineering advantages as discussed in the introduction section.

Satellite Image Classification

To compare the latency of the monolithic and modular solutions, the time that each solution takes to process 56700 data points is measured. The same is done with the distilled solutions. The latency measurements for each solution is taken on a machine with Intel(R) Xeon(R) CPU @ 2.20GHz, 13298580 kB of RAM and a Tesla P100 16280MiB GPU. The results are shown in fig. 3.10.

The latency results in fig. 3.10 shows that the non-distilled monolithic solution took 63.83% less time compared to the fastest modular solution to process the data load. However, distilled versions have been able to address this issue by achieving low latencies comparable to the mono-

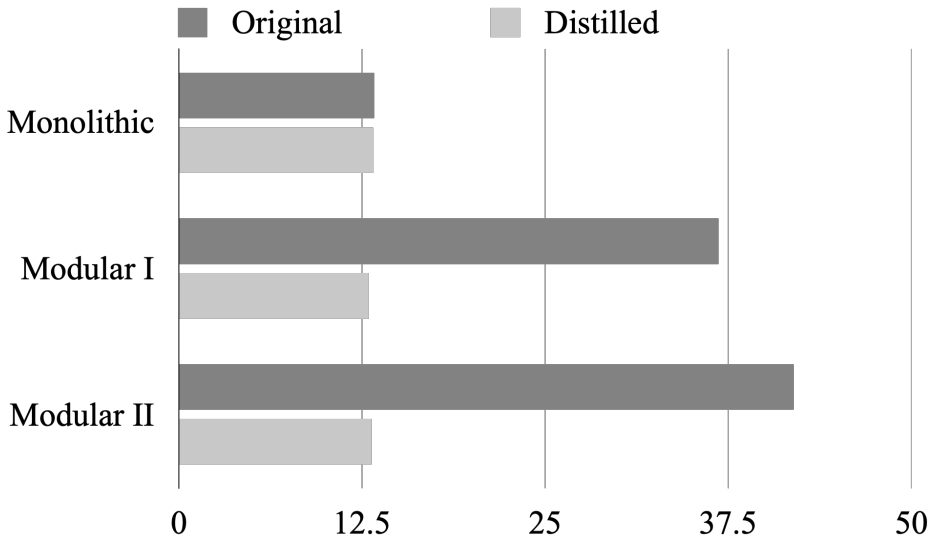

Figure 3.10: Cloudy satellite image classification latency (lower the better) of monolithic and modular solutions with the respective distilled versions.

lithic version, still having better accuracy than the monolithic version in the case of modular (II) solution.

3.3.3 Reusability

In this section we are testing whether an ML module that was used in one problem can be used in another problem. This is different from general transfer learning where base layers of a larger model are transferred to a similar task. Here we are reusing a semantically meaningful ML module in a different problem.

After the classification comparison, the cloud removal module is reused in a different task to evaluate the reusability of the module. In this task the RGB bands of the cloudy EuroSAT dataset are used to predict the NIR band for the image. Two monolithic and modular solutions are implemented for this task. The monolithic solution is trained end to end on the cloudy RGB

images. Monolithic model uses an encoder decoder network architecture similar to the one used in the cloud removal module but with a single output channel. The modular version predicts the NIR band in two stages. The first stage removes the clouds from training data by reusing the cloud removal module that was trained during the previous classification task. The second stage uses the output of the cloud removal module to predict the NIR band. Model used in the second stage uses the same network architecture used in the monolithic version. Two distilled models are created in this case as well using the monolithic and modular versions of the solution. Both modular and monolithic versions of the solutions are trained using the NIR prediction training data split that was explained before. The distillation was performed following the same process used for distilling the monolithic and modular (I) classification models. However, in this case the distilled models are pre trained with the NIR prediction dataset before the distillation. After the training and distillation steps, test split of the NIR prediction dataset is used to measure the accuracy of each solution. The results of this experiment are shown in fig. 3.11. The latency of each solution is measured for processing 56700 data points using each version of the solution on a machine with Intel(R) Xeon(R) CPU @ 2.20GHz, 13298580 kB of RAM and a Tesla P100 16280MiB GPU. The latency results are shown in fig. 3.12.

The results in fig. 3.11 and 3.12 resembles the pattern we observed in the classification case with monolithic and modular (I) solutions. The modular solution has higher performance in terms of accuracy/error. However, it has higher latency compared to the monolithic version as one would expect. The solution distilled from the modular model has lower error compared to the both original monolithic solution and the solution distilled from the monolithic solution while having comparable latency values.

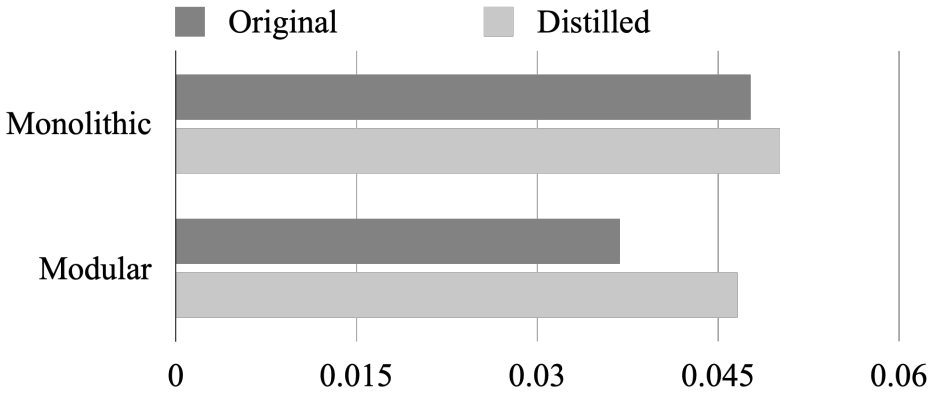

Figure 3.11: Mean squared error (lower the better) of monolithic and modular NIR prediction solutions including their distilled versions.

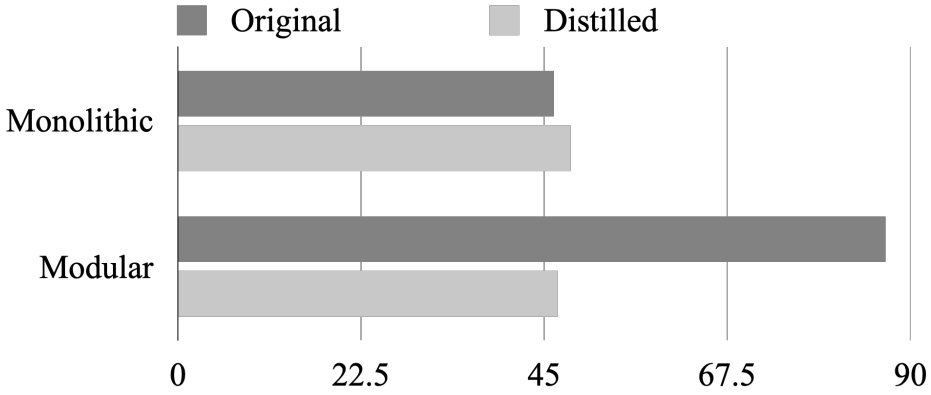

Figure 3.12: Latency (lower the better) of monolithic and modular NIR prediction solutions with the respective distilled versions.

3.3.4 Maintainability

In this section, we empirically highlight the trade-offs of the monolithic and the modular solutions with respect to maintainability as the requirements change. We will be considering a case where the requirements of the model change to handle noisy images in the cloudy satellite image classification problem.

In this experiment, we modify the classification dataset that we used before by adding gaussian noise to the RGB channels to represent noisy satellite images with cloud cover. Fig. 3.13 shows the images after adding noise. Then the noisy data is fed to the previously trained monolithic

Figure 3.13: Cloudy EuroSAT data points after adding gaussian noise.

classification model and the modular (I) classification model and the classification accuracy is measured using a held out test set. Additionally, we create an improved modular solution by updating the cloud removal module of the modular (I) solution. The cloud removal module is updated by training it with unlabeled noisy cloud images. This improvement makes the cloud removal module robust to noise. In the improved modular solution, the classification module remains to be the same module used in the original modular (I) solution. The accuracy of the improved modular solution is measured with the same noisy held out test data. Accuracy of each model is shown in fig. 3.14. As shown in the results in fig. 3.14, both monolithic and modular (I) solutions

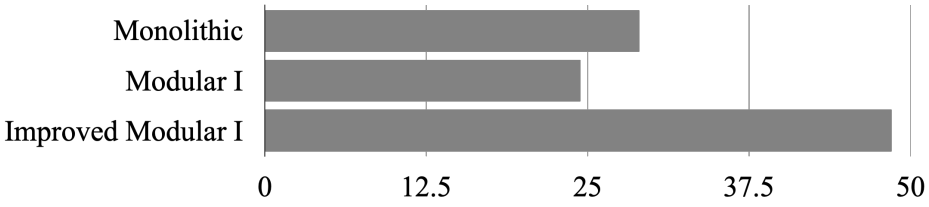

Figure 3.14: Classification accuracy for noisy satellite images with cloud cover.

significantly drop in accuracy when fed with noisy images. However, the modular (I) solution has the capacity to replace modules with improved ones. By replacing the cloud removal module with

an updated noise robust cloud removal module, the improved modular solution could achieve around 2 times higher accuracy compared to the original modular (I) solution. It should be noted that this accuracy gain was achieved without making any changes to the classification module and by only using noisy unlabeled data. The monolithic solution can not be improved this way using unlabeled data. Improving the monolithic solution needs labeled data that are usually labor intensive to produce.

3.4 Chapter Summary

In this work, while acknowledging the immense potential and positive impact of current deep learning technologies, we discussed the challenges and limitations of widespread monolithic deep learning technologies with respect to systems engineering concerns especially when it comes to wider adoption of these technologies in diverse organizations. We pointed out semantic modularity in machine learning as an interesting avenue to address a number of problems in this regard. Next, as a first step, we used three example problems to explore the benefits and trade-offs between developing machine learning solutions in a monolithic way and developing them in a multi-stage modular way. The experiments showed how modular solutions can reuse existing pretrained models and exploit more data to achieve higher accuracy and overcome data limitations in ways that monolithic solutions do not permit. Further, we used black box knowledge distillation to overcome the performance challenges that modular solutions can have and showed the impact of accuracy and latency in comparison to original monolithic and modular solutions. The experimental results in this work suggest that it is very interesting to further investigate the potential of

multi stage modular machine learning solution development in contrast to widespread monolithic end to end machine learning solution development.

Chapter 4

Towards Knowledge Graph-Enabled Machine Learning Service Description Framework[1]

4.1 Introduction

Developing a robust ML pipeline for any given application not only requires intricate knowledge about the domain but also significant expertise in the ML field. Building and managing effective ML-based solutions demands significant experience in various aspects such as *(1)* choosing appropriate ML technique (e.g., random forest, support vector machine, convolutional neural

[1]Major parts of this chapter appears in Menik and Ramaswamy, 2021

networks, etc.) and associated architecture (number of trees, number of layers, interconnection patterns, etc.), *(2)* finding and/or creating high quality training datasets, *(3)* feature engineering and hyper-parameter tuning, *(4)* deploying ML models on target computing infrastructures (cloud, cloudlets, end-devices, etc.) and *(5)* continuously managing and tuning the deployed services. Unfortunately, many domain experts often do not have this level of ML expertise, which has hitherto impeded wider adoption of ML in complex but important domains. Rapid advancement of the ML field has further aggravated this problem. Difficulty of converting work present in research literature into real world solutions due to the uniqueness of real world problems and reproducibility issues caused by missing descriptions also intensify the problem.

Machine Learning as a Service (MLAAS) seeks to address the above issue by enabling ML experts to expose ML functionalities as *services* which can then be shared, discovered, reused and integrated to create domain specific ML-driven applications Kim et al., 2018; Ribeiro et al., 2015; Soifer et al., 2019 by domain experts with limited ML background. Figure 4.1 shows an example of reusing and integrating ML services (OCR and Skew Correction) in two different ML application pipelines.

A robust and flexible ML service description that expresses the semantics of ML services [1] is critically important for designing efficacious search, composition, and reuse of ML services in MLAAS paradigm. ML model description framework provides a means for describing important information about ML models at the right abstraction level with embedded semantic information. By formalizing the manner in which ML services are described, it establishes consensus among practitioners, and it enables effective automation of search and discovery of ML services. These

[1] We use the term *ML service* in this paper to refer to ML models created and/or trained by ML experts in order to be used by others.

descriptions can be harnessed for building automated tools for checking the compatibility of individual services in a complex ML pipeline.

The overall objective of our research is to design a comprehensive description framework for ML services. Our approach aims to harness *knowledge graphs (KGs)* for ML service description. KGs are semantic-rich knowledge-based structures that organize entities, their properties, and their interrelationships in the form of large networks or graphs Paulheim, 2016. The contributions of this paper are as follows. 1. We present a requirements study of ML services that highlights the core properties of ML services that need to be incorporated in the description framework. 2. Based on the above requirements analysis, we lay out our vision of a novel five-dimensional KG-enabled ML service description framework with a specification template for each dimension. 3. We introduce unique conceptual formulations and novel variants of existing paradigms for supporting more cogent representation of the characteristics, constraints and inter-relationships of ML services. 4. We will be pointing out a number of interesting research directions and opportunities that stem from this paper.

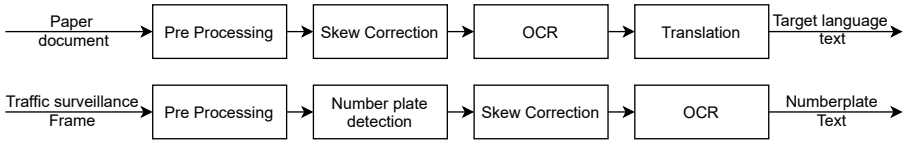

Figure 4.1: Two different ML pipelines that reuse an ML service.

4.2 Requirements Study

In this section we explore what aspects of ML services that need to be specified in order to enable various MLAAS functionalities. We have identified 5 major aspects that should be there in the

ML services description. 1. **Task Description**: This is a concise semantically annotated human readable description of the task that the ML service can perform. This enables domain experts to quickly understand the functionality of an ML service without expertise in ML. 2. **Input-Output (I-O) Description**: This aspect specifies the syntax and the semantics of the I-O formats of the ML service. This describes requirements, restrictions and constraints of the I-O of the ML service. As an example, an ML service that extracts land cover information from satellite images may require the input images to be of certain resolution or higher in RGB format and each pixel should represent a geographical area not more than $30m^2$. Similar restrictions and constraints are applicable to the output as well. This aspect of the description is important for checking the compatibility of an ML service to a real world problem/dataset. I-O description also helps composition and verification of ML pipelines using automated tools. 3. **ML Model Architecture Description**: This aspect contains information regarding an ML model itself. This includes the type of the ML model (e.g., random forest, SVM, Transformer, etc.) and the structural information of the ML model (e.g., components and their structure of a deep learning model, interconnections among components). Level of details to expose in this part of the description is up to the developer of the ML service depending on their business model etc. This aspect of the description helps users with expertise make more informed decisions when selecting ML services. Depending on the amount of information present this aspect of the description can enable platform independent code generation for ML services as well. 4. **Dataset and Training Description**: Dataset and the process of training have a major impact on the performance of a given ML model. Thus, this aspect describes the properties of the dataset that is used to train the ML service as well as the process used for training the ML model in concern. Users need this information to identify if a

given ML service is suitable for their situation. Also, this information is helpful when retraining the ML service with a different dataset if needed. 5. **Performance Description**: ML service users need an upfront indication of how a given ML service will perform on a given task. Thus, this aspect provides the information on the performance characteristics of the ML service. We identify two parts in this aspect of the description – **(a) accuracy description** that expresses how good the ML service is at a task. This includes accuracy percentage, precision, recall etc as well as evaluation protocol such as the number of folds, number of runs and the datasets used for accuracy evaluation. **(b) efficiency description** that reports characteristics of the model that capture aspects such as resource requirements/consumption, expected service throughput, expected latency etc. This information can be given as analytical models or as reports on evaluation runs with hardware and software configuration details. These aspects of the ML models are important for the consumers to decide if a given ML service is practical for a problem at hand with the resource and requirement constraints.

4.3 KG-Enabled ML Service Description Framework

In this section we present our KG-Enabled Machine Learning Service Description Framework. This framework describes the ML services in the form of a knowledge graph (ML-KG) that reuses concepts from existing knowledge graphs. Figure 4.2 shows the high level overview of an MLAAS platform that utilizes ML-KG. This platform enables, 1. ML service publishing, semantic searching/discovery and ML services composition. 2. ML model analytics capabilities utilizing the knowledge in ML-KG.

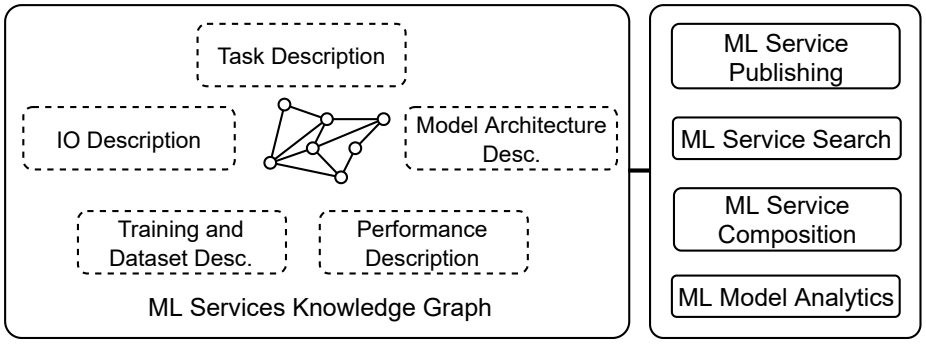

Figure 4.2: Overview of an MLAAS platform based on ML-KG.

4.3.1 Task Description

Task description is a brief natural language description of the main task that the ML service can perform. The keywords of the task description should be annotated with the entities in ML-KG. This has three main utilities. 1. Annotated description enables advanced semantic search capabilities. As an example annotations can be used to suggest similar ML services when exact matches are not available. e.g., If matching ML service is not available for the search "predict political bias of *tweets*" a system can use the annotations to find the relationship between *tweets* and *text* in ML-KG and suggest ML models that can predict the political bias of *text*. 2. Annotations in the description can be used to classify ML services into high level categories using the hierarchical relationships among the entities in ML-KG. Users can explore these categories in a drill down fashion to discover ML services. 3. When non-experts encounter unfamiliar terms, the annotations can help to locate the corresponding entity in ML-KG and learn more about it.

4.3.2 Input-Output (I-O) Description

Input to the ML service and the output of it should be described in terms of the syntax and the semantics. The syntax refers to the data structural descriptions (data formats and related constraints) of the input and output instances of the ML services. Semantics refers to the meaningful concepts, their relationships and constraints that are represented by the data structures. A description of these two sub-aspects of an input or an output instance of an ML service is referred to as an *I-O type*. All I-O instances of an ML service should comply with the I-O types. An ML service can be considered as a mapping from an input type to an output type. The I-O aspect of ML-KG is modeled by reusing the concepts present in existing knowledge graphs as shown in figure 4.3. This has three layers of information. 1. Entities and relationships in existing KGs. 2. I-O Types created based on existing KGs. 3. I-O aspect of ML services as mappings from input types to output types.

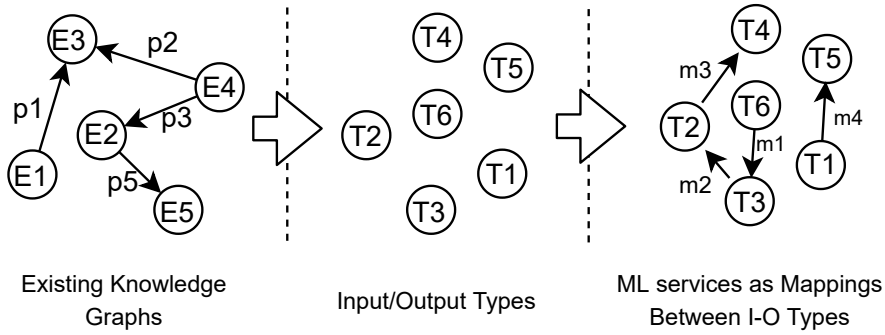

Figure 4.3: I-O aspect of ML-KG is constructed with three layers of information.

With the I-O aspect of ML-KG, ML service searching and composition can be posed as a path finding problem between I-O type entities in the graph. Here, when looking up an ML service for a problem, the input and the output of the problem should be described similarly to ML

service I-O types. Then, most related I-O type entities should be located in ML-KG and paths between them should be searched. There are 4 cases for this search result. 1. If the resulting path has a length of 1, an exactly matching ML service is available. 2. If the path length is greater than 1, several ML services have to be composed to create a solution. 3. If there are no paths between the located I-O types, the input and output constraints can be relaxed to find more I-O types and then look for paths among them. 4. If the I-O types can not be located in ML-KG, again the search constraints can be relaxed hoping to match more I-O types.

4.3.3 ML Model Architecture Description

In ML-KG, internal architectural details of the ML model of an ML service should be described in different levels of details. These levels of details correspond to the component composition hierarchy of an ML model. Highest level of details include relationships and categorizations in ML model level in a more abstract way. Lower levels contain details of basic components used in the ML model architectures. Figure 4.4 shows a partial visualization of the representation of internal details of ML models in ML-KG using a compound graph that has inclusion relationships. Internal details are represented in a hierarchical fashion that layers from top to bottom contain higher level details to lower level details. This aspect of the description serves several purposes. 1. Searching ML models with certain properties using subgraph matching queries. e.g., find all models with LSTM cells. 2. Generating the implementations for the ML models using the descriptions when sufficient information is present for a given ML model. 3. Depicting ML models with the assistance of tools for educational and communication purposes. 4. Using as a database of basic ML components that can be used in new ML models by the experts.

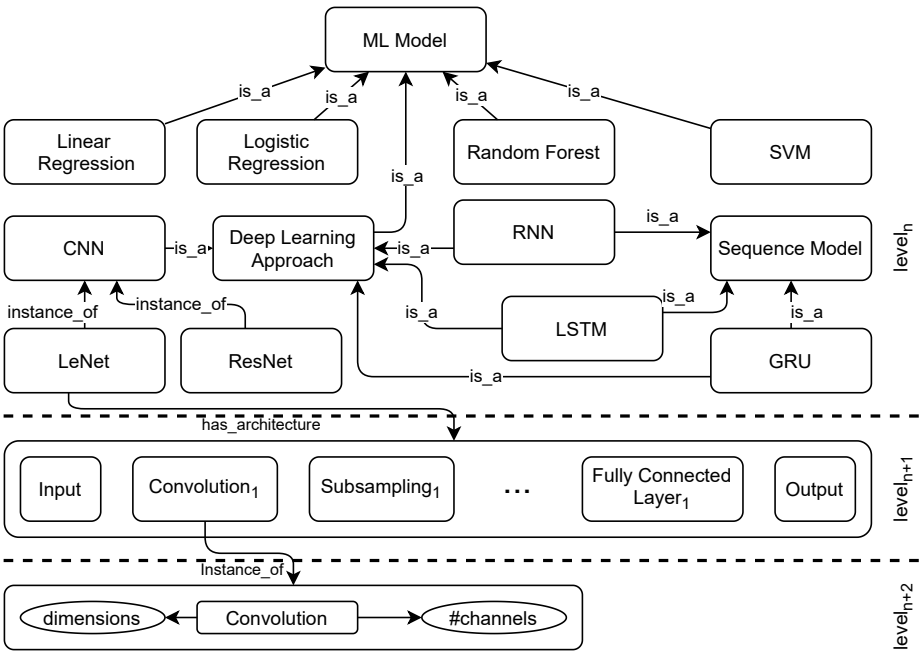

Figure 4.4: Partial visualization of the representation of internal architectural details of ML models in ML-KG.

4.3.4 Dataset and Training Description

Dataset makes a major impact on the performance of any ML solution. We will be focusing on statistical properties and the semantic and syntactic types of the datasets [2]. Most of the common statistical properties pertaining to datasets can be specified as numerical and categorical dataset properties of ML services in ML-KG. Semantic and syntactic type information of the data points in the dataset can be specified in a way very similar to the I-O type descriptions. The rest of the training description includes applicable training algorithms and hyper-parameters of the ML model as further dataset properties of ML services in ML-KG.

[2]There are a number of other properties in datasets that could contribute to the differences in the performance directly or indirectly. Some of these are very challenging to be described under a general formalization.

4.3.5 Performance Description

As discussed in the requirements study section, the performance description contains accuracy related and efficiency related performance metrics. These metrics can be standard or results of custom experiments and evaluations. The standard metrics can be associated with ML services as properties similar to the way discussed in the previous section. Custom metrics can also be associated with ML services with custom properties and sub graphs.

4.4 Chapter Summary

Machine learning as a service (MLAAS) has the potential to propel development and proliferation of innovative ML-driven applications in many important domains including agriculture, health and smart communities. For the MLAAS paradigm to be effective, it is imperative to build a robust, flexible and extensible ML services description framework. Towards this end, this paper presented an approach that harnesses the semantic-richness of knowledge graphs (KGs) for ML service description. We presented a unique five dimensional, KG-enabled service description framework that incorporates structures for specifying salient properties of ML services such as ML task description, input-output description, ML model architecture, dataset and training description and their performance characteristics.

Chapter 5

Modular Deep Learning with Softswitches

Currently there are several prominent ways of developing deep learning models. One is to create a custom deep learning architecture by composing several of many deep learning operators like convolutions, fully connected layers, batch norms etc. Another prominent way of developing a deep learning model is to make use of transfer learning approaches Zhuang et al., 2020. This can either be done by borrowing an existing architecture and training from the beginning or by borrowing a suitable set of pretrained base layers and customizing the model by adding additional operators and fine tuning the model by training it on a problem specific dataset. The first approach does not utilize existing work beyond the basic deep learning operators. The transfer learning approaches start building the new model on top of existing work. However, the way transfer learning reuses existing work is not a seamless process. In the first version of transfer learning that was mentioned above, the developers have to find an existing model architecture that best suits the

problem at hand and train a model from the ground up. In the second version of transfer learning, the developers have to find a set of base layers with a suitable architecture that was trained with a similar suitable dataset that matches the current problem. Finding such an architecture or a set of matching pretrained base layers is a trial and error process. This process is different and more challenging compared to how component reuse works in conventional software engineering.

As an alternative to the prevalent methods of developing deep learning models, we introduce a compositional modular deep learning architecture that results in a final model that contains modules with high level semantic meaning. These modules can later be reused as meaningful pretrained high level components to solve different problems in a way closer to how component reuse works in conventional software engineering. This architecture enables reusability of pretrained neural networks in a more fine grained and ready to use manner while enabling benefits of modularity that includes ability to update modules, replace modules with different characteristics. Further, this architecture is more transparent leading to improved debuggability compared to conventional monolithic deep learning models due to the semantic substructure. This architecture also enables the ability of domain expert intervention to improve model performance, more flexible and scalable distributed deployment abilities.

This architecture uses a module that we call a softswitch to compose a set of smaller neural network modules to develop a model for a higher level solution. The softswitch and the set of modules works with a representation produced by an encoder module that was trained in a self supervised manner and has a matching decoder that enables more model inspection and debugging abilities. We conduct a number of experiments with several benchmark datasets using the proposed architecture. The experimental results show that this architecture has the ability to

bring benefits of modularity to supervised classification problems with minimal trade-offs and comparable performance characteristics.

5.1 Modular Architecture with Softswitches

The proposed architecture has three main parts. First part of the architecture has an encoder module. This encoder module takes the input and produces an interpretable representation for the input. In this process, the encoder compresses the input instances to a more manageable representation. This helps to minimize the size of the overall modular architecture. The output produced by the encoder is provided as the input to the softswitch and a set of modules as shown in fig. 5.1. The output of the softswitch and the set of modules are used to produce the final output of the overall model.

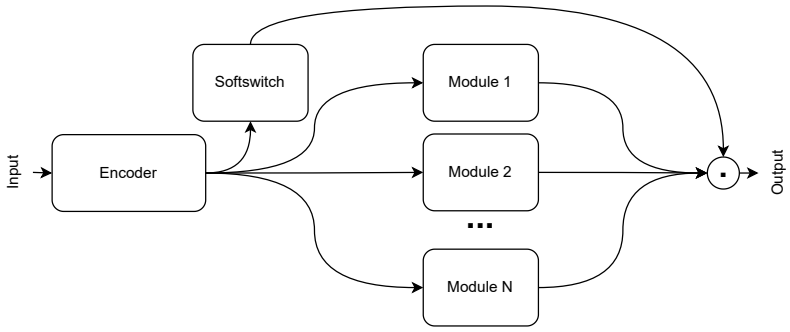

Figure 5.1: Proposed high level architecture that promotes modularity.

5.1.1 Input Encoder

In this architecture the encoder is a pretrained model that accommodates the universe of discourse for potential inputs for any module configuration that comes after the encoder. With this, the

encoder is intended to be a general component that is usable in different problem settings. The encoder is trained in a self supervised manner with unlabeled data that are less costly and easy to obtain. These type of universe of discourse models that are trained in a self-supervised manner with large amounts of unlabeled data are common in today's world with the advent of large language models and other self-supervised image processing models. However, the encoder that we use in this work is different from them because the universe of discourse for the models that we are interested in this work are generally smaller in comparison.

In our experiments we train an autoencoder architecture with both encoder and the decoder parts and later remove the decoder part to create the encoder model. The removed decoder model is kept aside for later use for transparency and debuggability of the final composed architecture that will be created with the encoder module. The autoencoder architecture that was used with the MNIST dataset in our experiments is shown in fig 5.2. In this case, the autoencoder architecture was created as a convolutional neural network. In fig 5.2, the left part of the image shows the encoder and the right part of the image shows the decoder.

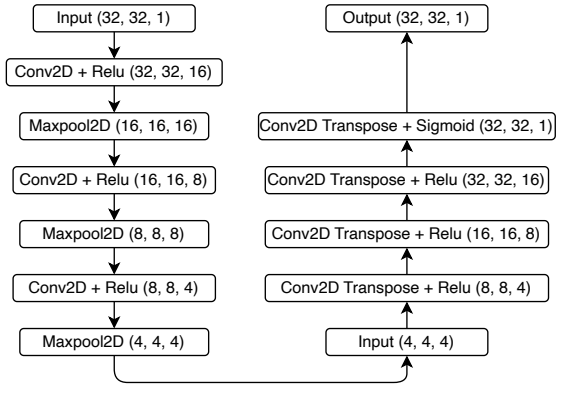

Figure 5.2: Autoencoder architecture that is used to train the encoder for MNIST dataset.

5.1.2 Softswitch Component

This component is similar to another classifier that learns to select the right module to minimize the loss function of the overall problem. The softswitch takes the encoder output as the input and produces a probability distribution using a softmax to select the right module to perform the classification. Softswitch learns from the dataset of the overall problem. Fig.5.3 shows the architecture of the softswitch that was used in one of the MNIST dataset experiments. The softswitch architecture shown here is able to compose two modules.

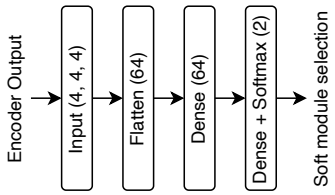

Figure 5.3: Softswitch architecture that is used for an MNIST composition.

5.1.3 Modules

Each module in the architecture is responsible for handling a subset of the classes in the overall problem. Each module takes the output of the encoder to produce the output of the subset of the classes that the module is responsible for. The architecture of the individual modules depends on the problem at hand. As an example the MNIST experiments presented in a later section use the dense architecture shown in fig. 5.4 for the modules. The module shown here is specialized in two classes.

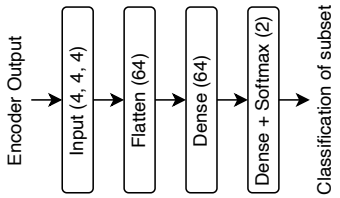

Figure 5.4: The architecture of each module that was used for an MNIST composition.

5.1.4 Model Output

The output of the overall model is the output of all modules weighted by the module selection probabilities that are predicted by the softswitch. This arrangement provides a direct path for the gradients to propagate to the softswitch and the modules during the training phase while enabling an uninterrupted direct connection from the encoder to the module inputs. Having an unmodified connection from the encoder to the modules is important for enabling module reuse.

5.2 Engineering Concerns

5.2.1 Module Reuse

The proposed architecture works by dividing the problem into a set of submodules with clear high level semantic meaning. These modules can be reused in other problems/models without any retraining or with minimal fine tuning. The modules for the new problem/model can come from more than one existing model as long as they are all based on the same encoder or a compatible encoder. When modules are reused the softswitch has to be trained to compose the modules

to solve the new problem. During the training time, this is done by freezing the weights of the encoder and the modules and leaving the weights of the softswitch open for changes. This training is done with the dataset that is available for the overall problem. See fig. 5.5 for a representation of this process. Here the module 1 from problem 1 and module 4 from problem 2 are reused to produce a solution for problem 3. See 5.2.5 for a discussion about module coupling and updating the encoder to handle a bigger scope when reusing modules.

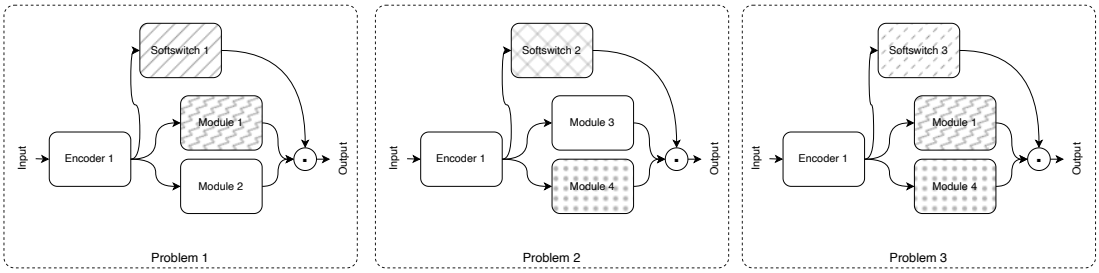

Figure 5.5: Visualization of reusing modules from two existing solutions to create a new solution.

5.2.2 Module Updates

Modules with high level meaning in the proposed architecture allows replacing modules with different characteristics without having to update the complete model. This is an important requirement in conventional software engineering. In conventional software engineering, separation of concerns, loose coupling and high cohesion are very important to manage complexity and improve maintainability. Following the same principles, the proposed architecture allows updating specific individual modules by freezing the weights of other modules as updates become necessary for different high level concerns. Further, the modules can also be replaced with other compatible pre-trained modules that are trained to work with the same encoder. See 5.2.5 for a discussion about module coupling.

5.2.3 Domain Expert Intervention

The way that the modules are broken down to solve the problem affects the engineering concerns and the performance characteristics of the overall solution. One can argue that this introduces an additional level of complexity to model development. However, the decisions one has to make here are more high level compared to low level model fine tuning decisions one typically has to make when developing deep learning models. Further, the resulting modules are reusable in other contexts. Being able to modularize the solution in different ways provides an additional dimension of flexibility when developing solutions. The module breakdown can be done in two ways. One is to subjectively decide the best separation of concerns for the modules using one's domain expertise. The other is to analytically decide the scope of each module. As an example, unsupervised clustering can be used to identify the clusters in the representation domain. Then the clusters that are close in proximity can be handled by separate modules.

5.2.4 Transparency

In conventional deep learning models a number of low level operators like linear units and are tightly connected to create a large network. These networks do not have a semantically meaningful substructure. Regardless of that there have been a number of attempts to interpret the mechanics and the decision making process of prevalent deep learning models Fan et al., 2021. However, the intrinsic monolithic nature of conventional models impacts the effectiveness of interpretability when it comes to explaining the mechanics of the model. Conversely, the proposed architecture has high level modules that specialize in subproblems of the overall problem. This introduces

semantic substructure to the model. This makes it possible to trace the path of input instances and explain the process that leads to the output of each input instance in a straightforward way. Further, the parts of the network have specific responsibilities associated with them. This enables troubleshooting similar to a conventional software solution in a modular way. Individual components in the network can be verified on their own and map issues to specific modules in order to solve them. As an example if a specific input is not producing the expected outcome, the first point to check could be the encoder in order to verify if the encoder produces a good representation as expected. This can be done by using the pre-trained decoder component of the autoencoder that the encoder was derived from. The next point to check could be the softswitch to examine if it chooses the correct module to produce the output for the given input. This can be done by examining the probability distribution of the softswitch for the given input. Finally the module that is responsible for the given input could be inspected to determine if it is producing the correct output for the given input instance. After the issues are mapped to specific components in the model, they can be resolved by updating the right component independently while the other parts of the model remain intact.

5.2.5 Encoder Updates

As discussed before the encoder is expected to be able to handle the universe of discourse for the set of problems that the encoder supports. See section 5.1.1 for more details on the encoder. The robustness of the encoder is crucial for the entire architecture because all the other components rely on the representation generated by the encoder. The softswitch and the set of modules use the output of the encoder for training and inference. This creates a coupling between the encoder and

the other components and this makes updating the encoder challenging. Updating the encoder with new training data can alter the representation for currently supported input instances and break the coupling between the encoder and the other modules leading to lower performance of the overall model. However, this problem can be mitigated by regularizing the output of the encoder with the existing representations that the encoder produces during the training process of updating the encoder.

5.3 Experimental Evaluation

We performed experiments with the proposed architecture using MNIST Deng, 2012 and Fashion-MNIST Xiao et al., 2017 datasets. Table 5.1 shows information about the datasets and the sizes of training, validation and testing splits used throughout the experiments. During the experiments the encoder module of each modular solution is pre-trained with the training data split without using any labels in a self supervised manner. The encoder is trained with an autoencoder with a bottleneck to reconstruct the input using mean squared error as the loss function.

Table 5.1: Dataset information

Information	**MNIST**	**Fashion-MNIST**
Classes	10	10
Datapoints	70,000	70,000
Training %	71%	71%
Validation %	15%	15%
Testing %	14%	14%

ResNet He et al., 2016 and a CNN architecture is used as the baseline in our experiments. None of these architectures are modular according to our definition. However, since these are widely used architectures with very good performance characteristics, they serve as a baseline to

understand where the proposed modular architecture lands in terms of performance characteristics. For the ResNet architecture we use the Keras implementation of ResNet Chollet et al., 2015. The other baseline CNN architecture we used has a convolutional base and a set of dense units connected to it to perform classification. The architecture that we used here is the best performer in terms of accuracy for the datasets we used out of 10 similarly sized architectures that we created. During each experiment, the baseline models are trained with the same amount of data as the other models with early stopping enabled. Early stopping has a tolerance of maximum three continuous epochs of higher loss than previous loss on the validation split. Training is stopped if the threshold exceeds and the best performing version out of model out of the epochs so far are taken for the experiment results.

5.3.1 Accuracy

Accuracy of the proposed architecture is evaluated by creating modular architectures with varying number of module counts and varying amounts of module scopes. Further, the accuracy is separately evaluated in the case where modules are trained from the beginning and in the case where modules are reused from existing solutions.

Evaluation of modules trained from the beginning for a given problem is conducted by creating three modular configurations. We name these three configurations composition 1, composition 2 and composition 3. Composition 1 uses two modules in the compositions. Composition 2 uses 3 modules and composition 3 uses 5 modules. Each composition divides the class space into equal partitions such that each module gets a partition to specialize in. Remaining classes after dividing the class space into equally sized partitions are ignored. Individual classes that are in each

partition are decided randomly. Each composition is trained with random class assignments to each module as explained for 3 times. For each time, the baseline models are also trained for the same class space and the same amount of data. The average accuracy values of each composition on a held out test set are reported in fig 5.6 and fig. 5.7. The accuracy results indicate that the modular solutions have comparable accuracy to the baseline solutions. Further, the accuracy values remain comparable regardless of the number of modules and the specialization of the modules.

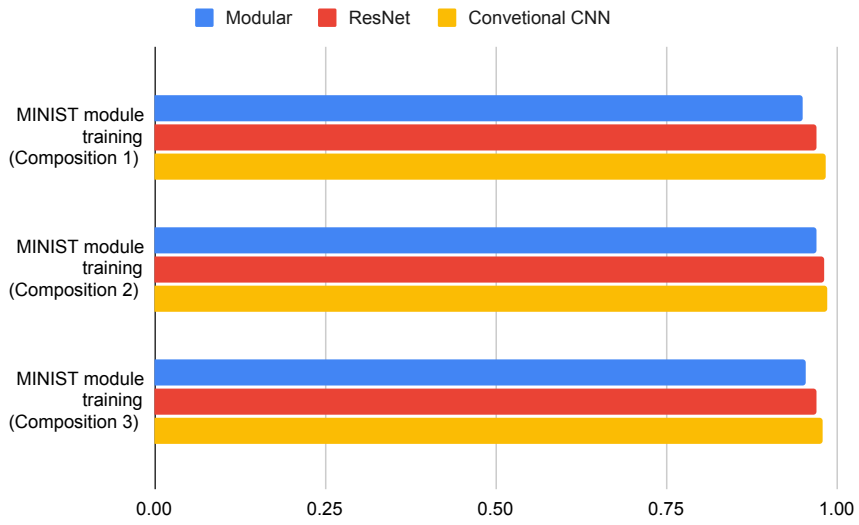

Figure 5.6: The accuracy of modular solutions when modules are trained for the problem compared to ResNet and CNN baselines with MNIST.

Evaluation of module reuse is performed by reusing the modules trained in the module training experiment to create different modular compositions. For this purpose, we create three other module compositions: composition 1, composition 2 and composition 3. Each of these compositions are created by reusing two to three modules trained in the module training experiment. Composition 1 in this experiment reuses one module from composition 2 and two modules from composition 3 from the module training experiment. Composition 2 of this experiment uses the

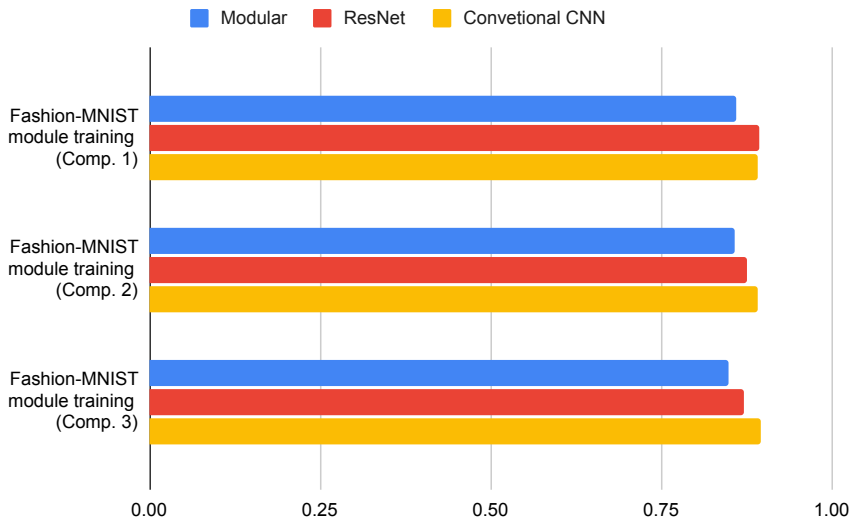

Figure 5.7: The accuracy of modular solutions when modules are trained for the problem compared to ResNet and CNN baselines with fashion-MNIST.

same structure as composition 1 in this experiment but with different molecules from the module training experiment. Both composition 1 and composition 2 in this experiment have modules with roughly equal scope, each module is responsible for two to three classes from the class space. Composition 3 in this experiment uses one module from composition 1 and one module from composition 3 from the module training experiment. Since composition 1 in the module training experiment has just two modules and composition 3 in the training experiment has 5 modules, The scope of modules in the two compositions are different. One module is responsible for five classes and the other is responsible for two classes. This results in a composition with an unbalanced scope for composition 3 in this experiment. This gives us an opportunity to see if the scope imbalance affects the effectiveness of the overall modular solution. The three compositions that are created by reusing the modules from the module training experiment are trained along with

the ResNet and CNN baseline models using the same amount of appropriate data. It should be noted that the weights of the reused modules were kept intact in the modular solutions and are not updated during the training. The experiment is performed 3 times with different module assignments with non-overlapping scope while preserving the composition structure that was discussed before and the average test accuracy is reported in fig. 5.8 and fig. 5.9. The results indicate that the modular solution can reuse modules without updating their weights and still achieve a comparable accuracy to the monolithic solutions. Further, accuracy of the modular solutions are not impacted by the imbalance of the scope of the modules that are reused in the solution.

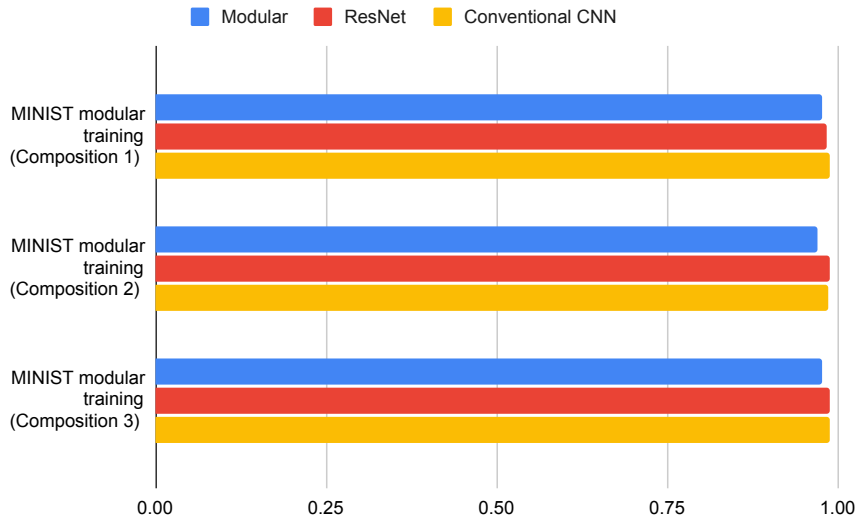

Figure 5.8: The accuracy of modular solutions when modules are reused as they are compared to ResNet and CNN baseline with MNIST.

5.3.2 Efficiency

The proposed modular architecture can be used to develop solutions by either training modules from the beginning or by reusing suitable modules from existing solutions. Fig. 5.10 shows the

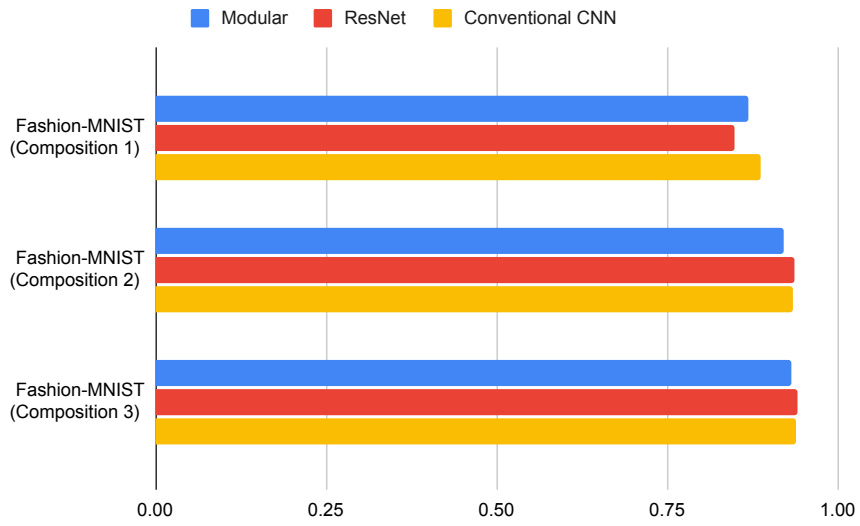

Figure 5.9: The accuracy of modular solutions when modules are reused as they are compared to ResNet and CNN baseline with Fashion-MNIST.

number of parameters that are trained with labeled data in the case where all modules are trained from the beginning in comparison to the case where all modules are reused from existing solutions. For this, we use the composition 1, composition 2 and composition 3 that was used in the module training experiment. The same compositions were recreated by reusing all of the modules to calculate the number of parameters trained with labeled data when all modules are reused without retraining them. It should be noted that the corresponding compositions that are created by training modules from the beginning and by reusing modules have comparable accuracy levels. The results from the fig. 5.10 indicate that reusing modules significantly cut down the number of parameters trained while achieving similar levels of accuracy.

The inference latency of the proposed architecture is measured by using the three compositions described in the module training experiments along with the ResNet and the other CNN

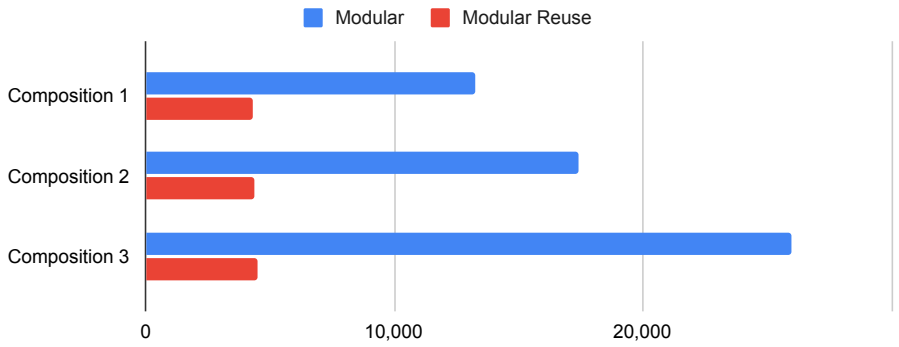

Figure 5.10: Number of parameters trained for a given modular composition when all modules are trained compared to when all modules are reused.

models. For this experiment, each module was used to predict 1500 batches of data with a batch size of 32. Then the time spent to complete all predictions is measured in seconds. The experiment is performed 7 times for each model and the average time spent is reported in fig. 5.11. The latency measurements for each solution is taken on a machine with Intel(R) Xeon(R) CPU @ 2.20GHz, 13298580 kB of RAM and a Tesla P100 16280MiB GPU. The results indicate that the proposed modular architecture has comparable latency to the baseline solutions while taking significantly less time for prediction compared to the baseline ResNet implementation.

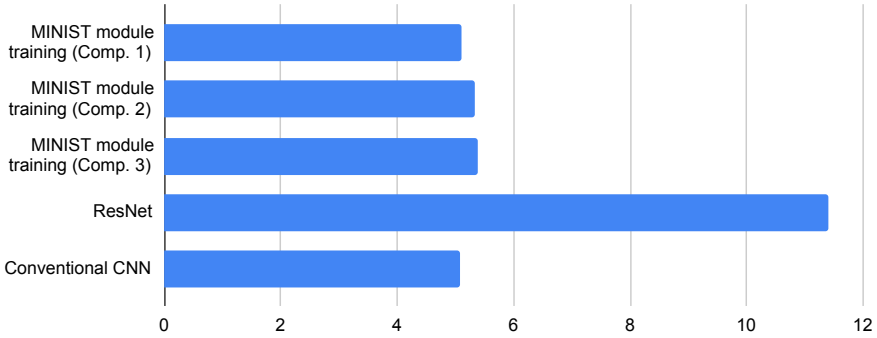

Figure 5.11: Inference time spent by the modular solutions compared to the baseline models.

Chapter 6

Summary and Future Work

This dissertation explored modularity in machine learning particularly focusing on deep learning models and studied the benefits of modularity in machine learning with regards to solution engineering. In essence the aim of this research is to reduce the burden of machine learning model development, deployment and maintenance by introducing modularity to machine learning solutions and make the technologies more accessible to a wider audience and a wider range of applications.

This work explored the solution engineering limitations of prevalent monolithic machine learning models and pointed out how these limitations can be addressed through modular machine learning solutions. The study also provided experimental evidence on how modular solutions can bring in a number of solution engineering advantages and data advantages while maintaining comparable core performance characteristics such as accuracy and latency in comparison to monolithic solutions. The findings emphasized the potential of modular machine learning solutions and highlighted further research that leads to broader adoption.

Furthermore, we have investigated the concept of machine learning as a service (MLAAS) and the need for a semantic-rich and flexible ML service description framework. By leveraging knowledge graphs (KGs), we have proposed a novel five-dimensional KG-enabled ML service description framework that incorporates major aspects of ML services. This framework provides a semantic approach to describe ML tasks, input-output specifications, model architectures, datasets and training details and performance characteristics. Such a framework is crucial for the realization of a modular machine learning ecosystem that democratizes AI and facilitates the publishing, searching, composing, and deploying of ML models.

Finally, we explored a new approach to develop deep learning models that results in models that contain semantically meaningful sub-structure with high-level meaning as opposed to the low level substructure that is present in prevalent deep learning models. The proposed architecture enabled a number of solution engineering advantages that includes module reusability, module updates and replacement, transparency, debuggability and domain expert intervention for performance optimization while maintaining comparable accuracy and efficiency characteristics to the baseline models.

Overall, this research highlights the significance of modular machine learning solutions and the importance of incorporating modularity and semantically meaningful sub-structure in machine learning models. The findings of this dissertation contribute to machine learning solution engineering and machine learning operations. It is our hope that this work will inspire further research and development in the pursuit of more cost effective, approachable, accessible and intuitive machine learning solutions and engineering methods.

6.1 Open Challenges and Future Work

6.1.1 Modular Machine Learning

Modularity in ML has a number of advantages. We discussed them in detail qualitatively and quantitatively. However, modular ML has several main open challenges when it comes to developing effective modular ML models.

Breaking down the original problem into a set of subproblems is not always feasible. Some problems do not have semantically meaningful subproblems. In these cases we have to consider the problem as an atomic unit and utilize end to end learning methods. After that the trained model has the potential to be used in a future Modular ML model.

In situations where new models have to be trained to solve subproblems, finding datasets for them can sometimes be challenging. This issue can be mitigated by performing the problem decomposition to match data that is available. In this case, the trade-offs of different problem decompositions should be further studied. Further, synthetic data can be helpful to fill some of the data gaps.

In some cases, module compositions may not perform in synergy as expected. Sometimes even if the individual models perform well, when they are composed together to solve a larger problem, the performance can be unexpectedly low. This can happen due to various incompatibilities among submodules. Studies on ML adversarial attacks may be able to shed some light in this regard. Further understanding the reasons for such failures and methods is important to make modular ML applicable in a wider array of problems.

Lack of feature rich repositories to publish and search ML models and datasets is another challenge to using modular ML in practice. Existing systems for publishing and searching ML models do not provide sufficient metadata and semantically meaningful search capabilities to look for models and datasets that can satisfy specific requirements. More advanced ML models and data repositories with semantic search capabilities and metadata support should be developed to make modular ML practical.

Addressing these challenges through future work is key to reaping the benefits of modular machine learning.

6.1.2 Machine Learning Module Description

We identify several future research works when it comes to describing machine learning modules/services using knowledge graphs. 1. Availability of a large number of existing ML models is key to the success of the ML description KG. Thus, scalable methods should be explored to rapidly grow ML-KG. 2. Further studies should be conducted to introduce sophisticated methods to describe semantic types and determine the degree of similarity between types since it is crucial for the ML service search/discovery, composition and related tools. 3. Methods that can predict the performance characteristics of ML service pipelines from the properties of individual ML services that are used to build the pipelines should be studied to be able to effectively compare the search results that are derived from ML-KG. 4. When ML-KG is populated with a large number of ML services, it becomes a large knowledge base of structured information on ML models. Ways to exploit this knowledge using data analytics techniques for the further advancement of ML theory and applications should be studied.

6.1.3 Large Language Models

Recent advancements in large language models are revolutionizing the field of machine learning and artificial intelligence. They are showing remarkable emergent capabilities in handling wide array of general intelligent tasks Wei et al., 2022 such as generating human-like text, writing poetry, summarizing text, generating programs given a description, predicting the output of a function given a description with an appropriate prompt. These capabilities enable unprecedented automations across various domains. However, a majority of large language models are monolithic. They do not have the semantically meaningful substructure that we explored in this dissertation. Because of this, despite the remarkable potential of these models, they still have the limitations of lack of modularity that we discussed in the previous chapters. Yet, these generative models can be a meaningful sub-part of a higher level modular solution. Since the language models can be fine-tuned or prompt engineered to imitate arbitrary functions, it is possible to decompose a high level solution into a set of sub-modules and define the communication protocols for the sub-modules. Then, the sub-modules can be created using large language models to follow the defined communication protocol through fine-tuning and/or prompt engineering. By doing this, we can enable partial transparency to complex solutions via the interfaces that the sub-modules interact with even in a case where the entire solution is developed using generative black box models. The solutions created this way resemble solutions that are developed following the microservices paradigm. This is very interesting because this approach has the potential to emulate complex systems with little to no code. Furthermore, the level of abstraction for sub-modules can be adjusted as per design requirements. If sub-modules are made more concrete (lower level), the final solution will

likely be more interpretable. However, this could increase both runtime and development time overheads. On the other hand, increasing the level of abstraction (making sub-modules higher level) likely leads to the opposite outcome: less interpretability, but potentially decreased runtime and development time overheads. Given the potential of this approach, it is interesting to further study the role of large language models in modular deep learning.

Bibliography

Ai hub | google cloud. (2020, March 3). Retrieved March 21, 2020, from https://cloud.google.com/ai-hub

Ai hub | google cloud. (2021, March 30). Retrieved March 30, 2021, from https://cloud.google.com/ai-hub

Ai platform | google cloud. (2020, September 25). Retrieved September 25, 2020, from https://cloud.google.com/ai-platform

Andreas, J., Rohrbach, M., Darrell, T., & Klein, D. (2016). Learning to compose neural networks for question answering. *arXiv preprint arXiv:1601.01705*.

Bai, J., Lu, F., Zhang, K., et al. (2019). Onnx: Open neural network exchange.

Bergstra, J., & Bengio, Y. (2012). Random search for hyper-parameter optimization. *Journal of machine learning research*, *13*(2).

Cases, I., Rosenbaum, C., Riemer, M., Geiger, A., Klinger, T., Tamkin, A., Li, O., Agarwal, S., Greene, J. D., Jurafsky, D., et al. (2019). Recursive routing networks: Learning to compose modules for language understanding. *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, 3631–3648.

Chard, R., Li, Z., Chard, K., Ward, L., Babuji, Y., Woodard, A., Tuecke, S., Blaiszik, B., Franklin, M., & Foster, I. (2019). Dlhub: Model and data serving for science. *2019 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 283–292.

Chollet, F., et al. (2015). Keras.

Csordás, R., van Steenkiste, S., & Schmidhuber, J. (2021). Are neural nets modular? inspecting functional modularity through differentiable weight masks. *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. https://openreview.net/forum?id=7uVcpu-gMD

Deng, L. (2012). The mnist database of handwritten digit images for machine learning research [best of the web]. *IEEE signal processing magazine*, *29*(6), 141–142.

Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Devlin, J., Chang, M., Lee, K., & Toutanova, K. (2019). BERT: pre-training of deep bidirectional transformers for language understanding. In J. Burstein, C. Doran, & T. Solorio (Eds.), *Proceedings of the 2019 conference of the north american chapter of the association for computational linguistics: Human language technologies, NAACL-HLT 2019, minneapolis, mn, usa, june 2-7, 2019, volume 1 (long and short papers)* (pp. 4171–4186). Association for Computational Linguistics. https://doi.org/10.18653/v1/n19-1423

Enomoto, K., Sakurada, K., Wang, W., Fukui, H., Matsuoka, M., Nakamura, R., & Kawaguchi, N. (2017). Filmy cloud removal on satellite imagery with multispectral conditional generative adversarial nets. *2017 IEEE Conference on Computer Vision and Pattern Recognition*

Workshops, CVPR Workshops 2017, Honolulu, HI, USA, July 21-26, 2017, 1533–1541. https://doi.org/10.1109/CVPRW.2017.197

Fan, F.-L., Xiong, J., Li, M., & Wang, G. (2021). On interpretability of artificial neural networks: A survey. *IEEE Transactions on Radiation and Plasma Medical Sciences*, *5*(6), 741–760.

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep Residual Learning for Image Recognition. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Helber, P., Bischke, B., Dengel, A., & Borth, D. (2019). Eurosat: A novel dataset and deep learning benchmark for land use and land cover classification. *IEEE J. Sel. Top. Appl. Earth Obs. Remote. Sens.*, *12*(7), 2217–2226. https://doi.org/10.1109/JSTARS.2019.2918242

Hinton, G. E., Vinyals, O., & Dean, J. (2015). Distilling the knowledge in a neural network. *CoRR*, *abs/1503.02531*. http://arxiv.org/abs/1503.02531

Huang, G., Liu, Z., van der Maaten, L., & Weinberger, K. Q. (2017). Densely Connected Convolutional Networks. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Isola, P., Zhu, J., Zhou, T., & Efros, A. A. (2017). Image-to-image translation with conditional adversarial networks. *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*, 5967–5976. https://doi.org/10.1109/CVPR.2017.632

Jamshidi, P., Pahl, C., Mendonça, N. C., Lewis, J., & Tilkov, S. (2018). Microservices: The journey so far and challenges ahead. *IEEE Software*, *35*(3), 24–35.

Keung, P., Lu, Y., Szarvas, G., & Smith, N. A. (2020). The multilingual amazon reviews corpus. In B. Webber, T. Cohn, Y. He, & Y. Liu (Eds.), *Proceedings of the 2020 conference on empirical methods in natural language processing, EMNLP 2020, online, november 16-20, 2020*

(pp. 4563–4568). Association for Computational Linguistics. https://doi.org/10.18653/v1/2020.emnlp-main.369

Kim, H., Kim, M., Seo, D., Kim, J., Park, H., Park, S., Jo, H., Kim, K., Yang, Y., Kim, Y., et al. (2018). Nsml: Meet the mlaas platform with a real-world case study. *arXiv preprint arXiv:1810.09957*.

Komer, B., Bergstra, J., & Eliasmith, C. (2014). Hyperopt-sklearn: Automatic hyperparameter configuration for scikit-learn. *ICML workshop on AutoML*, *9*, 50.

Li, L., & Talwalkar, A. (2020). Random search and reproducibility for neural architecture search. *Uncertainty in Artificial Intelligence*, 367–377.

Li, L., Jamieson, K. G., DeSalvo, G., Rostamizadeh, A., & Talwalkar, A. (2017). Hyperband: Bandit-based configuration evaluation for hyperparameter optimization. *ICLR (Poster)*.

Lin, J., & Zhang, J. (2013). A fast parameters selection method of support vector machine based on coarse grid search and pattern search. *2013 Fourth Global Congress on Intelligent Systems*, 77–81.

Liu, H., Simonyan, K., & Yang, Y. (2018). Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055*.

Machiraju, S., & Modi, R. (2018). Azure cognitive services. In *Developing bots with microsoft bots framework* (pp. 233–260). Springer.

Maclaurin, D., Duvenaud, D., & Adams, R. (2015). Gradient-based hyperparameter optimization through reversible learning. In F. Bach & D. Blei (Eds.), *Proceedings of the 32nd international conference on machine learning* (pp. 2113–2122). PMLR.

Meng, X., Bradley, J., Yavuz, B., Sparks, E., Venkataraman, S., Liu, D., Freeman, J., Tsai, D., Amde, M., Owen, S., et al. (2016). Mllib: Machine learning in apache spark. *The Journal of Machine Learning Research*, *17*(1), 1235–1241.

Menik, S., & Ramaswamy, L. (2021). Towards a robust knowledge graph-enabled machine learning service description framework. *15th IEEE International Conference on Semantic Computing, ICSC 2021, Laguna Hills, CA, USA, January 27-29, 2021*, 104–107. https://doi.org/10.1109/ICSC50631.2021.00026

Menik, S., & Ramaswamy, L. (2023). Towards modular machine learning solution development: Benefits and trade-offs. *CoRR*, *abs/2301.09753*. https://doi.org/10.48550/arXiv.2301.09753

Model zoo - deep learning code and pretrained models for transfer learning, educational purposes, and more. (2020, September 24). Retrieved September 24, 2020, from https://modelzoo.co

Nural, M. V., Peng, H., & Miller, J. A. (2017). Using meta-learning for model type selection in predictive big data analytics. *2017 IEEE International Conference on Big Data (Big Data)*, 2027–2036.

Papazoglou, M. P., & Van Den Heuvel, W.-J. (2007). Service oriented architectures: Approaches, technologies and research issues. *The VLDB journal*, *16*, 389–415.

Parnas, D. L. (1979). Designing software for ease of extension and contraction. *IEEE transactions on software engineering*, (2), 128–138.

Paulheim, H. (2016). Knowledge Graph Refinement: A Survey of Approaches and Evaluation Methods. *Semantic Web*.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., & Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, *12*, 2825–2830.

Publio, G. C., Esteves, D., Ławrynowicz, A., Panov, P., Soldatova, L., Soru, T., Vanschoren, J., & Zafar, H. (2018). Ml-schema: Exposing the semantics of machine learning with schemas and ontologies. *arXiv preprint arXiv:1807.05351*.

Ribeiro, M., Grolinger, K., & Capretz, M. A. (2015). Mlaas: Machine learning as a service. *2015 IEEE 14th International Conference on Machine Learning and Applications (ICMLA)*, 896–902.

Ronneberger, O., Fischer, P., & Brox, T. (2015). U-net: Convolutional networks for biomedical image segmentation. In N. Navab, J. Hornegger, W. M. W. III, & A. F. Frangi (Eds.), *Medical image computing and computer-assisted intervention - MICCAI 2015 - 18th international conference munich, germany, october 5 - 9, 2015, proceedings, part III* (pp. 234–241). Springer. https://doi.org/10.1007/978-3-319-24574-4_28

Sagi, O., & Rokach, L. (2018). Ensemble learning: A survey. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, *8*(4), e1249.

Soifer, J., Li, J., Li, M., Zhu, J., Li, Y., He, Y., Zheng, E., Oltean, A., Mosyak, M., Barnes, C., et al. (2019). Deep learning inference service at microsoft. *2019 {USENIX} Conference on Operational Machine Learning (OpML 19)*, 15–17.

Stanley, K. O., & Miikkulainen, R. (2002). Evolving neural networks through augmenting topologies. *Evolutionary computation*, *10*(2), 99–127.

Suganuma, M., Shirakawa, S., & Nagao, T. (2018). A genetic programming approach to designing convolutional neural network architectures. In J. Lang (Ed.), *Proceedings of the twenty-seventh international joint conference on artificial intelligence, IJCAI 2018, july 13-19, 2018, stockholm, sweden* (pp. 5369–5373). ijcai.org. https://doi.org/10.24963/ijcai.2018/755

Tan, M., & Le, Q. V. (2019). Efficientnet: Rethinking model scaling for convolutional neural networks. In K. Chaudhuri & R. Salakhutdinov (Eds.), *Proceedings of the 36th international conference on machine learning, ICML 2019, 9-15 june 2019, long beach, california, USA* (pp. 6105–6114). PMLR. http://proceedings.mlr.press/v97/tan19a.html

Tiedemann, J. (2020). The Tatoeba Translation Challenge – Realistic data sets for low resource and multilingual MT. *Proceedings of the Fifth Conference on Machine Translation*, 1174–1182. https://www.aclweb.org/anthology/2020.wmt-1.139

Touvron, H., Cord, M., Douze, M., Massa, F., Sablayrolles, A., & Jégou, H. (2021). Training data-efficient image transformers & distillation through attention. *International conference on machine learning*, 10347–10357.

Tsay, J., Braz, A., Hirzel, M., Shinnar, A., & Mummert, T. (2020). Aimmx: Artificial intelligence model metadata extractor. *Proceedings of the 17th International Conference on Mining Software Repositories*, 81–92.

Vanschoren, J. (2018). Meta-learning: A survey. *arXiv preprint arXiv:1810.03548*.

Vanschoren, J., van Rijn, J. N., Bischl, B., & Torgo, L. (2013). Openml: Networked science in machine learning. *SIGKDD Explorations*, *15*(2), 49–60. https://doi.org/10.1145/2641190.2641198

Vartak, M., Subramanyam, H., Lee, W.-E., Viswanathan, S., Husnoo, S., Madden, S., & Zaharia, M. (2016). Modeldb: A system for machine learning model management. *Proceedings of the Workshop on Human-In-the-Loop Data Analytics*, 1–3.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). Attention is all you need. *Advances in neural information processing systems*, *30*.

Wei, J., Tay, Y., Bommasani, R., Raffel, C., Zoph, B., Borgeaud, S., Yogatama, D., Bosma, M., Zhou, D., Metzler, D., Chi, E. H., Hashimoto, T., Vinyals, O., Liang, P., Dean, J., & Fedus, W. (2022). Emergent abilities of large language models. *Trans. Mach. Learn. Res.*, *2022*. https://openreview.net/forum?id=yzkSU5zdwD

Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., Funtowicz, M., & Brew, J. (2019). Huggingface's transformers: State-of-the-art natural language processing. *ArXiv*, *abs/1910.03771*.

Wu, Y., Schuster, M., Chen, Z., Le, Q. V., Norouzi, M., Macherey, W., Krikun, M., Cao, Y., Gao, Q., Macherey, K., Klingner, J., Shah, A., Johnson, M., Liu, X., Kaiser, L., Gouws, S., Kato, Y., Kudo, T., Kazawa, H., … Dean, J. (2016). Google's neural machine translation system: Bridging the gap between human and machine translation. *CoRR*, *abs/1609.08144*. http://arxiv.org/abs/1609.08144

Xiao, H., Rasul, K., & Vollgraf, R. (2017). Fashion-mnist: A novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*.

Xie, L., & Yuille, A. L. (2017). Genetic CNN. *IEEE International Conference on Computer Vision, ICCV 2017, Venice, Italy, October 22-29, 2017*, 1388–1397. https://doi.org/10.1109/ICCV.2017.154

Yang, Z., Dai, Z., Yang, Y., Carbonell, J., Salakhutdinov, R. R., & Le, Q. V. (2019). Xlnet: Generalized autoregressive pretraining for language understanding. *Advances in neural information processing systems*, *32*.

Zhuang, F., Qi, Z., Duan, K., Xi, D., Zhu, Y., Zhu, H., Xiong, H., & He, Q. (2020). A comprehensive survey on transfer learning. *Proceedings of the IEEE*, *109*(1), 43–76.

Zoph, B., & Le, Q. V. (2017). Neural architecture search with reinforcement learning. *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. https://openreview.net/forum?id=r1Ue8Hcxg