

Deep Learning in Elixir: An Investigation

11785 Fall 2024 - Group 57

Darren Siegel

Abstract

Deep learning has revolutionized machine learning workflows, but its adoption outside Python's ecosystem remains limited due to the lack of mature tooling in alternative languages. This project investigates the state of deep learning within the Elixir programming language ecosystem, which has traditionally been known for building fault-tolerant, distributed systems rather than machine learning applications. The study evaluates Elixir's capabilities by developing, training, and deploying deep learning models using its emerging tools: Nx, Axon, Bumblebee, and LiveBook.

Three distinct efforts were undertaken. First, a "from-scratch" linear model for MFCC-based phoneme classification was implemented, with inputs being MFCC features and outputs being phoneme class predictions. This task assessed Elixir's performance compared to Python for basic model training. Second, a transfer learning-based image classifier was developed using a pre-trained ResNet-50 model fine-tuned to classify web accessibility issues, specifically detecting "good" images versus screenshots of source code or tabular data. The inputs were uploaded images, and the outputs were categorical classifications to guide authors toward more accessible alternatives. Finally, the trained classifier was deployed in a production Elixir application, Torus, to automate the identification of inaccessible images in an online course authoring platform.

This work highlights both the promise and limitations of Elixir for deep learning. While Elixir's tools demonstrate potential for production deployments, challenges such as inefficient data collation, limited documentation, and the absence of robust community support hinder broader adoption. Addressing these limitations and integrating standards like ONNX for model interoperability could position Elixir as a competitive alternative in specialized domains, particularly those requiring scalable and reliable deployment environments.

Introduction

This project set out to assess the state of Deep Learning within the Elixir programming language ecosystem by designing, training, and deploying deep learning models using only the libraries and tools available in that language's ecosystem. The objective is to explore whether Elixir's recent advancements make it a competitive alternative to Python for building and deploying machine learning, and in particular, deep learning solutions.

Elixir, traditionally known for its strengths in building fault tolerant, reliable distributed systems has recently undergone significant improvements in its machine learning capabilities with the

introduction of libraries and tools such as Nx, Axon, Bumblebee, and LiveBook. Nx serves as an analog to Numpy, Axon is a framework for building and training neural networks, Bumblebee provides pre-trained models for a variety of tasks, and LiveBook offers an interactive notebook environment akin to Jupyter. These tools signal the growing maturity of the Elixir ecosystem as a viable option for deep learning.

Through this project, I aimed to evaluate Elixir's capabilities for deep learning, not only in terms of model performance but also in ease of integration, deployment, and scalability in production systems. In doing so, I compared Elixir's emerging ecosystem to Python's mature deep learning tools and provided an informed assessment of the practicality of adopting Elixir for deep learning in industry settings.

My investigation consisted of three separate efforts:

Part A: “From scratch” model development: First, to directly compare the Elixir and Python deep learning development experience, I replicated my homework 1, part 2 submission entirely in Elixir. This provided an opportunity to assess the capabilities of Nx and Axon against Numpy and PyTorch, respectively, for training models “from scratch”. I sought to not only replicate the functionality of the PyTorch based training - but also the efficiency as well, so careful attention was paid to achieving optimizations similar to that of the DataLoader. In addition, I based the entire implementation in LiveBook in order to vet its capabilities and developer experience against that of Jupyter style notebooks (Google Colab is what I used for my HW1PT2).

Part B: Transfer Learning based Image Classifier: This project resulted in an application of transfer learning to fine-tune an existing pre-trained vision model for a domain specific task. The model that I developed will help mitigate a common web accessibility problem seen in online courses: the inclusion of screenshots of source code and screenshots of tabular information instead of including these materials as actual embedded source code and as actual HTML tables. This essentially is a problem of accessibility - a screenshot of source code isn't readable by a screen reader.

Part C: Deployment of Model for Inference In a Production Elixir Application:

The third part of the project resulted in deploying the model from Part B into Torus, an Elixir based open-source course authoring and delivery system developed here at Carnegie Mellon. The model was integrated into the Torus system as an end user facing feature. The integration of this model into Torus, beyond being an exploration of what it takes to deploy models into an Elixir-based system in production, contributes to improving accessibility in educational materials.

Literature Review

The model development portion of this project aims to identify images uploaded by course authors into the Torus platform (for display to students) which violate web accessibility guidelines.

1. Transfer Learning

In my project, I chose transfer learning to develop a 3-class image classifier to distinguish "good" images from screenshots of "source code" or screenshots of "tabular" content. This approach aligns with the fundamental objective of transfer learning, which aims to leverage knowledge from related domains (pre-trained models on large image datasets) to improve performance on a specific target task without requiring extensive labeled data in the target domain.

The problem here involves classifying web-based content in an online course authoring platform, where images are often uploaded as alternatives to HTML elements, particularly tables and code. This choice can negatively impact accessibility since HTML offers better support for screen readers and search engines. Using a transfer learning-based model addresses this by accurately classifying these images, helping authors understand when a more accessible format is preferable. Transfer learning, as highlighted in the paper, reduces the dependency on a large target dataset by using pre-trained models that generalize well, thus making it an ideal solution for my project where extensive, labeled examples of "good," "source code," and "table" images may not be readily available.

Transfer learning (Zhuang et al, 2020) also allows us to benefit from the rapid advances in machine learning models, including homogeneous approaches that adapt models trained on related image data, like general-purpose object recognition, to our specific context. This aligns with the survey paper's emphasis on selecting appropriate models to optimize practical performance in diverse application domains, ensuring my classifier can quickly and reliably handle course content uploads, ultimately promoting accessibility and enhancing the authoring process.

2. ResNet: Deep Residual Learning for Image Recognition

In building my 3-class image classifier, I use a ResNet-based model due to its foundational advancements in training deep networks, as described in the seminal paper "Deep Residual Learning for Image Recognition" (Kaiming et al, 2015). The ResNet architecture introduces a residual learning framework that enables significantly deeper models without the optimization difficulties that typically arise as depth increases. By structuring layers to learn residual functions relative to their inputs rather than learning entirely new representations at each layer, ResNet enables effective gradient flow through the network, which reduces issues like vanishing gradients and makes it easier to train deeper models effectively.

This framework is especially valuable in my project, where identifying nuanced differences among images—such as distinguishing between "good" images, screenshots of "source code," and "table" screenshots—requires a model with high capacity to capture subtle patterns. ResNet's architecture allows us to use deeper models that have been shown to generalize well across complex visual tasks, leveraging the network's ability to extract detailed, hierarchical features.

Empirical results from the ResNet paper underscore that deep residual networks not only achieve state-of-the-art performance on challenging datasets like ImageNet and COCO but also maintain lower computational complexity compared to similarly deep models. This makes ResNet a practical choice for integration into my online course authoring platform, where it can provide robust, high-quality image classification with manageable computational demands. The success of ResNet across multiple benchmarks solidifies its reliability for diverse applications, making it an ideal backbone for accurately classifying image types in course content to enhance accessibility.

Baseline model and description

Part A: “From Scratch” Model Development

This part reimplements our linear layer model for MFCC phoneme classification from homework 1, part 2, entirely in Elixir.

Part B: Transfer Learning based Image Classifier

The project will use a pre-trained image classification model available via Elixir’s “Bumblebee” library, which includes (amongst others) ResNet-50. A cross entropy loss is used, given the multi-class classification goal.

<https://huggingface.co/microsoft/resnet-50>

Baseline implementation completeness

Part A: “From Scratch” Model Development

Implementation Status:

Fully implemented and functional in an Elixir Livebook, reimplementing the linear model from Homework 1, Part 2, for MFCC phoneme classification.

<https://github.com/darrensiegel/11785-project/blob/master/hw1pt2.livemd>

- **Data access:** Raw data download and unarchiving, and file reading in DataSet module
- **Dataset Loading:** Successfully loads the dataset with an asynchronous DataLoader, supporting efficient data batching.
- **Data Augmentation:** Incorporates time and frequency masking for augmenting the dataset.
- **Training and Evaluation:** Established training and evaluation loop, performing the model training and validation processes.

Challenges and Limitations:

- **Data Access:** Downloading Kaggle data directly into Elixir posed challenges. Due to the lack of a direct HTTP download method from Kaggle, I uploaded the dataset to an S3 bucket for access.
- **File Unzipping:** Using LiveBook on Fly.io raised issues with unzipping Kaggle data, which the Erlang `:erlang.unzip` function could not handle. To work around this, I created a .tar.gz version for successful unpacking.
- **Numpy File Compatibility:** The `Nx.load_numpy!` function could read MFCC numerical data but not transcript files encoded as string arrays, as it only supports numeric Numpy arrays. To solve this, I authored a Python script to convert transcript data into integer Numpy arrays representing phoneme indices.
- **DataLoader Performance:** High training time (estimated at dozens of hours for 1 epoch!) is attributed to inefficiencies in the `get_item` method in the Dataset in conjunction with batch collation of the DataLoader. This issue arises because Elixir's immutable linked list structure lacks random, constant time, access, which affects performance despite attempts to mitigate it with first my own Nx-based "FastArray" module and an existing third-party "Arrays" module (<https://github.com/Qqwy/elixir-arrays>). The latter only guaranteed $O(\log n)$ element access, which drops off, inconveniently at around 25,000 elements.

Part B: Transfer Learning-based Image Classifier

Implementation Status:

Fully implemented and functional in an Elixir Livebook, utilizing transfer learning on a ResNet model (via Bumblebee) customized for a 3-class classification of images (good images, code screenshots, and table screenshots).

The loss function optimized here is categorical, cross entropy loss:

$$\mathcal{L}_{\text{CCE}} = -\frac{1}{N} \sum_{i=1}^N \sum_{k=1}^C y_{i,k} \log(\hat{y}_{i,k})$$

Where N is the number of samples, C is the number of classes, $y_{i,k}$ is the true label for sample i and class k $\hat{y}_{i,k}$ is the predicted probability for sample i and class k .

<https://github.com/darrensiegel/11785-project/blob/master/transfer-learning.livemd>

- **Dataset:** Manually created a dataset with around 130 images per class, split into 80/20 for training and development.
- **Data Loading:** Asynchronous DataLoader support is included.
- **Model Customization:** Loaded a pre-trained ResNet-50 model and customized it for this classification task.

- **Training and Evaluation:** Established a training and evaluation loop for iterating over the dataset and improving classification performance. I ran many different training runs at different train and validation splits, with different optimizers and different learning rates.
- **Key Metric:** Training and validation accuracy were the two key metrics that I focused on. I needed (obviously) to exceed 33% accuracy to ensure that I was simply seeing “random” results.

Part C: Deployment of Model for Inference In a Production Elixir Application:

The model trained in Part B was fully integrated into Torus, fronted by an end user facing feature in the application’s “Media Library”. As a course author uploads images to use in their learning pages, the system performs inference on the images to determine if they are screenshots of tabular data or source code.

The source code is available on this GitHub branch:

<https://github.com/Simon-Initiative/oli-torus/tree/11785>

A delta between the main branch and that branch can be seen here:

<https://github.com/Simon-Initiative/oli-torus/compare/master...11785>

Results

Part A: “From Scratch” Model Development

No results were obtained from this part of the project, as the per-epoch training time is seemingly on the order of dozens of hours due to the inherent performance limitations described above.

Part B: Transfer Learning-based Image Classifier

The best performing training run yielded 76% validation accuracy at 74% training accuracy. This was 11 epochs using SGD at 1e-3 learning rate.

Part C: Deployment of Model for Inference In a Production Elixir Application:

A set of fine-tuned weights was saved from Part B and the model was successfully deployed into Torus on an experimental branch. The project video demonstrates this integration and the end user facing feature which uses it.

Discussion

This project highlights the potential of Elixir for deep learning but also reveals its current limitations, particularly when compared to Python's mature ecosystem. Each part of the investigation presented unique insights into Elixir's capabilities and challenges.

Livebook Integration and Usability

One of the promising features of Livebook is its seamless integration with Fly.io, enabling quick attachment to remote GPUs for accelerated computation. This is a significant advantage for developers and researchers who need access to scalable computing resources. However, Livebook's overall usability was a mixed experience. Editing quirks, such as the cursor randomly jumping during code editing, detracted from the workflow. A more serious issue was the unintended re-execution of notebook cells, even when no changes were made. For instance, cells reloading 4GB of MFCC data unnecessarily consumed time and resources, creating frustration and inefficiency. These usability challenges make Livebook less polished than Python-based environments like Jupyter or Google Colab.

Axon's Strengths and Weaknesses

Axon demonstrated strong potential for building and training neural networks, particularly for simpler tasks. Its API is intuitive, and the integration with Nx enables effective tensor operations. However, as complexity increased in Part B, such as during transfer learning and model customization, the limited documentation and examples became a significant hurdle. In Python, the abundance of tutorials, community-driven content, and extensive library documentation makes troubleshooting and research trivial by comparison. This gap in support represents a major barrier to adoption for Elixir as a deep learning platform, especially for non-experts.

Performance Issues in Batch Collation (Part A)

The performance of Elixir in collating batches during Part A was the most significant disappointment of the project. The inherent limitations of immutable data structures in Elixir and the absence of a robust solution for random access proved to be a bottleneck. Despite attempts to optimize the process—including concatenating all MFCCs into a single Nx tensor—the training loop's inefficiency rendered Part A impractical. By contrast, Python's PyTorch DataLoader easily handles large datasets and batch processing with minimal overhead, showcasing its superiority in these fundamental operations.

Broader Implications

These challenges reflect the early-stage development of Elixir's deep learning ecosystem. While tools like Axon, Nx, and Bumblebee have made substantial progress in enabling neural network development within Elixir, there is still a significant gap in terms of performance, usability, and community support when compared to Python. For Elixir to become a viable alternative for deep

learning, it will need further optimization, improved documentation, and broader community engagement.

Bright Spots and Opportunities

Despite the challenges, there are bright spots. The ability to integrate the model into Torus demonstrates Elixir's strength in combining deep learning with production systems, particularly in domains where its fault tolerance and distributed capabilities are advantageous. Livebook's GPU integration also shows promise, providing an infrastructure that could become competitive with more refinement. These features indicate potential areas for growth and investment in Elixir's deep learning tools.

Future Work

Part A: "From scratch" model development

Investigating and fixing the underlying performance issues and the resulting limitations would be the most important part of "future work" were I to revisit this in the future.

Part B: Transfer Learning based Image Classifier

While this part was functionally complete, constructing an even larger image dataset could improve model performance.

Part C: Deployment of Model for Inference In a Production Elixir Application

Future work for this area involves monitoring the evolving support for the ONNX standard within the Elixir ecosystem. As ONNX matures in Elixir, it could enable seamless interoperability by allowing models trained in Python to be exported to ONNX format and deployed directly in Elixir applications. This advancement would bridge the gap between Python's rich training ecosystem and Elixir's strengths in production deployment.

Conclusion

In summary, while Elixir shows promise for integrating deep learning into production environments and handling distributed workloads, its current limitations in ease of use, performance, and ecosystem maturity hinder its adoption for more traditional machine learning workflows. However, with continued development and community engagement, Elixir could grow into a valuable tool for deep learning in niche applications.

Division of work

I was the sole member of this project team; I did 100% of the work.

GitHub Repositories

Parts A and B: <https://github.com/darrensiegel/11785-project>

The Torus changes: <https://github.com/Simon-Initiative/oli-torus/compare/master...11785>

Citations

Fuzhen Zhuang, Zhiyuan Qi, Keyu Duan, Dongbo Xi, Yongchun Zhu, Hengshu Zhu, Hui Xiong, & Qing He. (2020). A Comprehensive Survey of Transfer Learning

Kaiming He, Xiangyu Zhang, Shaoqing Ren, & Jian Sun. (2015). Deep Residual Learning for Image Recognition.