Typhon: Parallel Transfer on Heterogeneous Datasets for Cancer Detection in Computer-Aided Diagnosis

Giuseppe Cuccu
eXascale Infolab
University of Fribourg
Fribourg, Switzerland
giuseppe.cuccu@unifr.ch

Christophe Broillet
eXascale Infolab
University of Fribourg
Fribourg, Switzerland
christophe.broillet@unifr.ch

Carolin Reischauer
Department of Medicine
University of Fribourg
Fribourg, Switzerland
carolin.reischauer@unifr.ch

Harriet Thoeny
Department of Radiology
Cantonal Hospital of Fribourg
Fribourg, Switzerland
harriet.thoeny@h-fr.ch

Philippe Cudré-Mauroux
eXascale Infolab
University of Fribourg
Fribourg, Switzerland
philippe.cudre-mauroux@unifr.ch

Abstract*—We present Typhon, a new Deep Learning framework that trains a single model using multiple, heterogeneous datasets leveraging parallel transfer. This aims to improve the performance of Deep Learning methods in critical applications afflicted by data scarcity, such as computer-aided diagnosis for cancer detection, where large datasets are rare or unfeasible but many smaller datasets may be available. The key idea is to assemble sufficient data for training deep models by selecting a set of multiple, potentially smaller and heterogeneous datasets, as long as they all exhibit similar visual features, such as common with medical imaging applications. The Typhon model architecture is composed of a single Feature Extractor and multiple Decision Makers, in sequence but explicitly separated. The Feature Extractor is trained using all datasets with a focus on producing generic features which are useful across all datasets. The Decision Makers are each paired with a different dataset, and specialized to take decisions based on the output of the Feature Extractor. Our training method is based on the concept of parallel transfer: on each epoch, we train on just one batch from each dataset in turn. This is done by pairing the correct Decision Maker on top of the shared Feature Extractor, then training the resulting model end-to-end on the data batch using classical methods. The actual design is inherently more complex, as we had to overcome a set of major challenges such as dataset imbalance, moving target, catastrophic forgetting, and issues with initialization viability. Once made viable, however, this methods excels at strictly enforcing feature generalization and delaying or even preventing overfitting. We present our results on the widely adopted PROSTATEx MRI dataset for prostate cancer classification, using additional datasets of brain MRI and lung CT images to boost the model's performance. Typhon improves on our previous work based on sequential transfer (Hydra) by over 7%, which compounds to a 15% improvement over classical methods and 12% over transfer learning, while only seeing 54% more samples than classical end-to-end training on a single dataset.

978-1-6654-8045-1/22/$31.00 ©2022 IEEE

***Index Terms*—Data Scarcity, Parallel Transfer, Medical Imaging, Cancer Detection, Overfitting**

I. Introduction and motivation

Over the past decade, Deep Learning (DL) has revolutionized entire fields, such as natural language or image processing. Domains limited by *data scarcity*, however, still lag behind, as state-of-the-art method require increasingly more data to train. Some of these domains are, however, of crucial importance, such as computer-aided diagnosis of tumoral masses in medical images.

Creating large datasets in this field comes with significant challenges, mostly put in place explicitly to protecting patient privacy: releasing medical data for public usage is both an ethical and bureaucratic nightmare. Moreover, labeling such images requires the contribution of expert radiologists, a resource that is inherently very limited, as clinical work is typically prioritized over research efforts. Obtaining sufficient data for classical Deep Learning methods has proven an unresolved challenge for many years now. Our approach focuses instead into constructing a model and a training procedure that can excel using data that is already available: relatively small datasets, though potentially in larger numbers.

The first step in this challenge is to understand the root issue that causes Deep Learning to be so data-hungry: the *vanishing gradient* problem. While the last layers of a neural network are not difficult to train *per se*, as the errors can be computed fairly precisely against the labels, the first layers of the network only receive a residual gradient that has been back-propagated through all intermediate layers: the deeper the network, the more the loss of precision of this error signal,

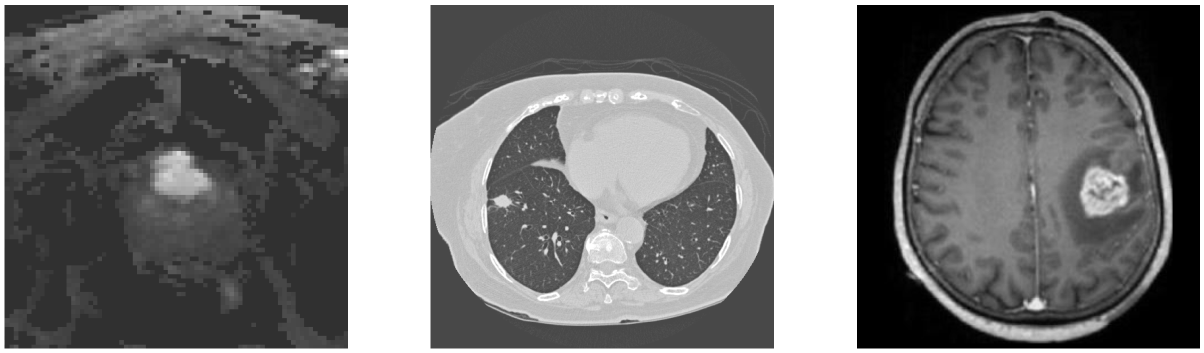

Figure 1: **Datasets.** Our Typhon framework trains a model on multiple, potentially heterogeneous, datasets. We present results on the PROSTATEx MRI (a) dataset, augmented by adopting the Lung CT (b) and Kaggle Brain (c) datasets for training.

and the more update steps and data points are consequently necessary to average out the its noise.

In most image processing applications, especially based on convolutional networks, these first layers have at the same time a very critical role: to extract increasingly complex visual features from the input image, eventually creating a meaningful high-abstraction *feature space*, on top of which a smaller set of layers, typically fully-connected feed-forward layers, can take the final decision. The first and larger part of the network, even in classic monolithic designs, implicitly acts as a *Feature Extractor* (FE), with the smaller fully-connected section on top defining instead a *Decision Maker* (DM). These parts are often clearly distinguishable in most modern architectures, although the differentiation is rarely leveraged explicitly.

Training the Decision Maker is typically not overly challenging: its size is smaller, and the error signal it receives is of higher quality simply because it is composed of the very last layers of the network. Training the Feature Extractor layers is what makes most Deep Learning methods data-hungry. Our initial inspiration for this work is that, if only there could be a way to train the Feature Extractor on multiple datasets, their (single) smaller size would be less of an issue, and the resulting feature space could be of higher-quality. This would in turn allow each of the Decision Makers to further improve their responses, even if trained only on one dataset each.

We started on this path with a framework named *Hydra* [1], which tried to address the problem from the perspective of *sequential transfer*, akin to Transfer Learning [2]. Hydra also adopts an architecture with one Feature Extractor and multiple Decision Makers, but trains on each dataset in turn until convergence, to then discard the past Decision Maker. The approach is repeated once for each support dataset, leaving the actual target dataset for last as a final *model specialization*. This approach already improved significantly over standard Transfer Learning methods; its convoluted training process, however, while showing the potential of the approach, still suffered from catastrophic forgetting and early training termination, due to how easy is for a large model to overfit on small datasets.

Since then, our team has been focusing on *parallel transfer* instead, a radically different approach initially locked behind a set of significant challenges (see Section II). This work shows the result of our efforts in this direction, presenting our new framework named **Typhon**.

A. Related work

The idea of reusing the feature extraction layers originally trained for another application is at the core of *Transfer Learning* [3], [4]: a model is initially trained on a larger dataset, then the decision-making layers are re-initialized and re-trained on a new task, without altering the original feature-extraction layers. Such a technique is extremely effective in applications where training a particularly large model on billions of data points is unfeasible or prohibitively expensive for most; but if one instance of such a model is trained once and made publicly available, Transfer Learning allows reusing its painstakingly well-trained feature extraction layers on a new application, allowing for producing effective models in a cost-effective manner.

Abubakar et al. [5] claim that the “Transfer Learning process is used in two approaches: fine-tuning, where some modifications are made and as an off-the-shelf feature extractor where features are extracted in order to train a machine learning classifier”. Our previous work, Hydra [1], alternated layer-freezing and fine-tuning steps over each dataset in turn. Hydra can thereby be seen as an asynchronous multi-task learning algorithm, where the datasets are provided in turn, switching between last layers training and whole model training. Samala et al. [6] showed that “multi-task Transfer Learning may be an effective approach for training DCNN in medical imaging applications when training samples from a single modality are limited”, which is our case. The association of a pre-trained model with multi-task learning demonstrated its efficacy [7]. Our approach keeps the essence of this architecture, but is fundamentally different in three ways: (1) we do not use a pre-trained model coming from another source (they use ImageNet); (2) we use multiple datasets of different body parts

Algorithm 1 Typhon training process

```
Inputs:
  $\Sigma$: Set of $n$ training datasets $\sigma_i$, accessed via a LoopLoader (see Section II-C)
  $\mathcal{M}_0$: Complete Typhon model (see Section II-A)
        This includes one shared Feature Extractor (FE) and $n$ dedicated Decision Makers (DM$_i$).
        Initialization is done with our custom bootstrapping using *all datasets*, see Table I for details.
Main:
  $\mathcal{M} \leftarrow \mathcal{M}_0$
  for each epoch $e_i$ do
      for $\sigma_i$ in $\Sigma$ do                                  ▷ For each epoch, loop through the datasets
          $b \leftarrow$ getBatch($\sigma_i$)          ▷ Obtain a batch; this re-loads and re-shuffles $\sigma_i$ as needed
          $\mathcal{M}\langle \mathrm{FE}, \mathrm{DM}_i \rangle \leftarrow$ train($\mathcal{M}\langle \mathrm{FE}, \mathrm{DM}_i \rangle, b$)      ▷ Train on one batch, affecting both FE and DM$_i$
  return $\mathcal{M}$
```

alternatively (multiple steps); and (3) each head of the multi-task learning is trained at a different step.

Our approach also relates to Multitask Learning [8]. Here the objective is also to learn a single model to address multiple *tasks*, which can correspond to different datasets as well as to different targets on the same one. However, the key difference is that Multitask Learning expects to activate the entirety of the model for each input, on all tasks: the inputs from each task are concatenated then passed together to the model's input as a single item. We adopt a more granular approach in our work, as we train selective portions of the network (selecting individual Decision Makers) at each step.

We test the performance of our framework in the context of Computer-Aided Diagnosis (CAD) of malignant lesions. As our target dataset, we selected the SPIE-AAPM-NCI Prostate MR Classification Challenge (PROSTATEx) dataset [9]–[11], arguably the most well-known publicly available dataset for prostate lesion classification, initially published in 2017 as part of the *PROSTATEx Grand Challenge* [12]. It includes multi-parametric MRI scans from 204 patients, including multiple planes and different sew, in DICOM format.

We base our architectural and data augmentation choices on the work by Song et al. [13], which – to the best of our knowledge – published the best on technique on the PROSTATEx datasets so far. Their work builds on the MRI-as-color-channels stacking intuition from the input data stacks images from XMasNet, but with a novel network architecture inspired by VGG [14]. We re-implement their network architecture and image augmentation procedure exactly to establish a performance baseline, before adapting it to our framework to produce our results.

In order to test our multi-dataset approach, we introduce two more datasets with different imaging modalities (CT scans and MRIs), body parts (lung and brain) and file types (PNG and JPEG): the SPIE-AAPM Lung CT Challenge dataset [15]–[17] (Lung CT), and the "Brain MRI Images for Brain Tumor Detection" dataset [18] (Kaggle Brain). A discussed above, state-of-the-art results rely on dedicated architectures (e.g. multi-channel 2D convolutions) that optimize the performance based on a single, homogeneous dataset (e.g. multiparametric MRI scans). We aim instead at leveraging heterogeneous data types and modalities, to enable selecting candidate datasets from a broader pool. To this goal, we utilize only one sequence from the PROSTATEx dataset: the Diffusion-weighted MR images with the highest b-value. While this choice can be considered an handicap in principle, it also allows us to match the visual features of other capture methods, our thesis being that this advantage will eventually overcome the initial hindrance, while providing fair comparison across dataset results.

B. Contributions

This paper presents the Typhon framework, a novel method to train a multi-headed neural network model on a set of (potentially heterogeneous and small) datasets, using parallel transfer. The similarities to our previous Hydra framework are barely aesthetic, as training the model using parallel transfer brings an entirely different sets of challenges, requiring the creation of entirely new training and even bootstrapping methods. This is further detailed in Section II, and led to the following contributions:

- A new, always-available multi-headed design for our model;
- A custom bootstrapping technique, without which parallel transfer was originally non-viable;
- A novel training procedure learning from all datasets in parallel, which embraces and even leverages catastrophic forgetting on moving targets, to boost model generalization;
- A data loading structure producing an uninterrupted stream of data batches, which tackles our novel problem of dataset size imbalance;
- A significant performance improvement over our previous results: 7% increase in AUC over Hydra, which compounds to 15% over classical learning methods;
- An entirely new open source reference implementation, professionally designed for ease of reproducibility, adop-

tion and extension[1].

II. Challenges and breakthroughs

Algorithm 1 presents the overall training process of Typhon. Create a viable parallel training method and routine required significant efforts: the next section highlights the main challenges we met, and how we addressed them.

A. Moving target: the Typhon model

Our previous work was based on sequential transfer: each dataset is learned thoroughly (until convergence/overfitting) before moving on to the next one. This still leaves the model liable to catastrophic forgetting: training on each subsequent dataset is a chance to overwrite potentially useful features learned from previous datasets, without a chance for recovery. This fundamental issue is side-stepped in this work by switching to parallel transfer: the datasets are constantly rotated at each epoch, after *a single batch of training*. There is no training of a single dataset until convergence; instead, after learning a bit from one batch, the learning objective is abruptly moved on to the next dataset. This escalates the catastrophic forgetting, into a problem of *moving target*, where the learning and convergence are hindered by the learning objective constantly switching between datasets. This is naturally expected to delay the learning process, if not grinding it to a halt altogether; Typhon however, leverages these problems instead to *improve* its performance. This begins with its new network model: at all times, from the very start of the training process, one "body" (the common Feature Extractor) and all "heads" (the Decision Makers) are constantly maintained and kept viable, rather than splitting and matching different parts sequentially as was done in Hydra.

The Feature Extractor is trained on a different dataset after each batch. Every time a bit is learned from one specific dataset, the training immediately switches to the next dataset, enforcing to forget any learned information that was **uniquely specific** to that first dataset. In this case, the second dataset will generate an error signal that will unlearn that first bit. But if instead that bit of information is instead sufficiently generic to be of used also by the second Decision Maker on the second dataset, it should be expected to improve the model's performance, not generating errors, and not being forgotten. In this case, the bit of learning is not lost but reinforced. Typhon leverages catastrophic forgetting to fundamentally purge all single-dataset overfitting from the training process at each dataset rotation. The only information retained through the learning is, by necessity, pertaining and useful to all datasets. This boosts Typhon's Feature Extractor's innate focus towards generalization. The result is a higher-quality feature space, which in turn supports better learning for the Decision Makers, and better results overall. We show this experimentally in Section IV.

[1]Reference implementation: https://github.com/eXascaleInfolab/typhon

B. Initialization: full-model bootstrapping

Using sequential transfer for Hydra meant that the initialization of the different Decision Makers could be delayed until the training reached the corresponding dataset. This allowed bootstrapping both Feature Extractor and one Decision Maker, as a standard end-to-end model, specifically for the target dataset. The remaining Decision Makers could then be adapted to the features that were produced through training on the previous datasets. The Typhon model, however, requires the Feature Extractor and all Decision Makers to be available from the very first epoch, as the whole model is used by the training process trains on one batch from each dataset in turn at each epoch. Starting with a completely random initialization was the biggest problem at the beginning of the project: if the random initialization does not produce somewhat sensible results from the first epoch, the moving target problem between randomized Decision Makers simply tears apart any further attempt to learn useful features, in turn ensuring that he Decision Makers themselves will never improve.

We address this problem with a new bootstrapping method based on Random Weight Guessing (RWG; [19]), where a set of candidate models are initialized with random weights in turn, then scored on each dataset's validation set. The resulting scores thus depend on the quality of the feature extractor, on the specific utility of the features for the particular dataset, and on the quality of the decision maker, as well as the compatibility of those specific features across all datasets. Each score measures the model affinity to its own optimization goal, making the score aggregation overall a complex multi-optimization problem. Scoring one model by simply averaging the scores across all datasets typically led to mediocre results, leading to an iterative process where we examined and ultimately discarded several options. Our best results, as presented in this paper, still look for a high average across all dataset scores, but also requires for at least two Decision Makers to have higher scores than other candidates. The rationale behind our choice is that, in order to get a meaningful model to bootstrap our training, the Feature Extractor is the most crucial part that needs to be of high quality. Having a high score on a single dataset could be the result of a good Feature Extractor paired with a good Decision Maker, or it could still be simply the result of overfitting. To avoid the second case, we simply look at the scores of the other Decision Makers over the other datasets: for at least a second score to be high, the feature space generated by the Feature Extractor needs a certain ability to generalize across the two datasets, which is everything we really need from the bootstrapping process to stave off the initial issues with the moving target. The third lower score can be attributed to bad luck in the Decision Maker generation, which will be fixed by the subsequent training.

C. Dataset imbalance: loop-loaders

The next challenge starts as soon as the datasets are loaded. At each epoch, Typhon requires one batch from each of the datasets. The smaller dataset will thus be exhausted before the others are thoroughly processed. Naively, this would either

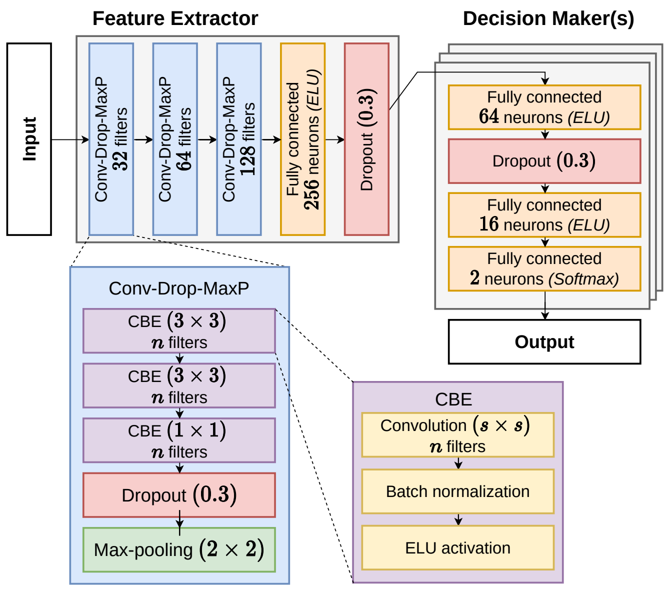
Feature Extractor
Decision Maker(s)
Input
Conv-Drop-MaxP 32 filters
Conv-Drop-MaxP 64 filters
Conv-Drop-MaxP 128 filters
Fully connected 256 neurons (ELU)
Dropout (0.3)
Fully connected 64 neurons (ELU)
Dropout (0.3)
Fully connected 16 neurons (ELU)
Fully connected 2 neurons (Softmax)
Output
Conv-Drop-MaxP
CBE (3 × 3) n filters
CBE (3 × 3) n filters
CBE (1 × 1) n filters
Dropout (0.3)
Max-pooling (2 × 2)
CBE
Convolution (s × s) n filters
Batch normalization
ELU activation

Figure 2: **Network architecture.** For fair comparison with our previous work [1], the network architecture is unchanged. The architecture is based on a custom VGG-16 by Song et al. [13]; we then split the layers explicitly into a Feature Extractor (top left) and three Decision Makers (top right), enforcing roles specialization.

require the training to stop, or to continue with one less dataset. Both these approaches are unsatisfactory because (i) useful information could be found in the remaining data of the larger datasets, but also (ii) continuing training with one dataset removed would expose its features to unbalanced catastrophic forgetting, which we were so careful in managing so far.

The problem is fundamentally comparable to *class imbalance* on a single dataset, where the naive mitigation of removing data from over-represented classes is also avoided in favor of generating variations of existing data for the under-represented classes (*data augmentation*). Still, this approach risks diluting the quality of the training set, as the synthetic data generated as variation of the initial data increases the dataset size without introducing novel information [20]–[24]. Typhon addresses this issue by allowing for the dynamic and asynchronous *reuse* of the available data of the smaller-sized dataset, making the fewer available points show up proportionally more often as the training progresses.

A special *loop-loader* data structure wraps each dataset loader in our reference implementation. A dataset is thus first loaded, randomly shuffled, and partitioned into batches of the required size. Each epoch utilizes one batch from the dataset. Once all batches are exhausted, our loop-loader triggers a re-shuffling and re-partitioning of the data, then transparently returns the first batch from this new set. As a consequence, unlimited new batches are always available regardless of dataset size.

An important consequence of this approach is that single data points belonging to smaller datasets will be seen more times (and more often) than a corresponding data point in their larger counterparts. This is typically avoided in the literature because imbalances quickly lead to overfitting on the smaller subset of data, which is a serious concern in applications leveraging sequential information transfer, which is the most common case. In Typhon however, we found out that this issue was entirely mitigated by our parallel transfer process and its constantly moving learning target. The smaller dataset remains relevant until the end of the training process: this ensures fair representation of the information it uniquely contains, while also constantly causing catastrophic forgetting on potentially overfitting features uniquely specific to the larger datasets.

This sums up the main challenges behind the design and implementation of Typhon, which can be described in more detail in the following section.

III. Method

Typhon is inspired by the same goal behind Hydra: to enable learning from multiple datasets, by separating feature extraction and decision making in the model. As discussed in Section II, however, our new training method based on *parallel transfer* prompted a radical overhaul of every single component of our framework, leading to an entirely different method, and consequently different results.

Only the data pre-processing is unaffected by the transition to parallel transfer, which allows us to use exactly the same data as our previous work [1]. This ensures a fair comparison of the performance between the two methods. The main assumption carried over remains that all datasets exhibit similar visual features, which can be expected in medical imaging even across formats and techniques. This is simply due to all images having, broadly speaking, the same subject: the human body internal workings, captured as images of density differentials, and focused on the detection of abnormal lesions.

A. Model design

Figure 3 shows an instance of the full Typhon model, with the shared Feature Extractor focusing on generalization, plus a set of Decision Makers specializing on single datasets, available at all times. For contrast, Hydra allows access to only one Decision Maker at a time, requiring a new one to be initialized and pre-trained when switching target dataset.

From a classic Deep Learning perspective, our separation of Feature Extractor and Decision Maker(s) is arbitrary. In principle, deciding which layer belongs to which part mostly follows convention (convolution plus pooling for feature extraction, fully-connected layers for decision making), though experimenting with different cut points allowed us to further improve on this decision. This means that in principle any desired network architecture available in the literature can be adapted to Typhon: the user simply needs to decide on a partition point, and generate multiple copies of the decision making layers. For fair comparison with our previous results however, our results are once again based on the same VGG-16 [14] architecture specialized by Song et al. [13] for prostate lesion classification, split after all convolution and pooling

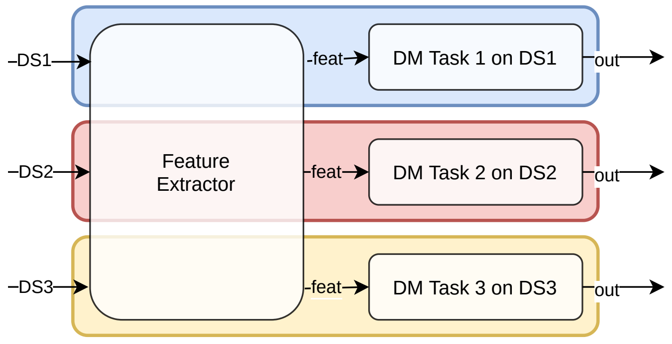

Figure 3: **Typhon model.** Three optimizers are instantiated for the training, each seeing only one end-to-end model (shared Feature Extractor plus one Decision Maker) running on one dataset (DS). At each activation, only one sample from one dataset is passed as input, activating the shared Feature Extractor and only one of the Decision Makers, corresponding to the specific dataset. For each epoch of training, each optimizers is used in turn on just one batch from one dataset; compounded by the tiny batch size favored by Typhon, epochs iterate much quicker than in classical learning. This is to further exacerbate the *moving target* issue coming from using parallel transfer, which Typhon leverages to further enforce generalization on the shared Feature Extractor.

plus one more layer of fully-connected neurons. The split is also kept at the same point as our previous work (see Figure 2), uniquely for fair comparison. Exploratory work shows undeniable potential for alternative architectures to further improve the results presented here (see Section VI-B), though this is left for future work.

As the Feature Extractor and all Decision Makers need to work at the same time and in concert from the very first epoch, naive random initialization may consistently produce parametrizations that are not viable for parallel transfer, as they get pulled apart by the moving target. This is addressed with a bootstrapping process using Random Weight Guessing (RWG; [19]) and a tailored evaluation function, as explained in Section II.

B. Training

With the bootstrapping method providing a meaningful starting point to start the learning process, the training can loop across all datasets already from epoch one. This drastically simplifies the Typhon training process with respect to our previous Hydra, as highlighted in Algorithm 1, which takes a form closer to classical (end-to-end) training. The two main differences are: (i) at each epoch, the model is trained on one batch from each of the datasets, and (ii) this requires the framework to switch across Decision Makers, based on the dataset, multiple times for each epoch.

C. Specialization

The whole design of Typhon revolves around maximizing the generalization capability of the Feature Extractor. Specialization is in principle entirely relegated to the Decision Makers, which are smaller in size anyway to prevent overfitting. Using heterogeneous datasets, features specific to only one dataset are typically discarded by the Typhon training process. This is by design also, as it allows squeezing all possible feature generalization from the datasets combination, but could as well likely limit the final performance on any one dataset.

To test this assumption, we follow up our training with a *specialization* step, akin to the last step in the Hydra training: the model is trained end-to-end on the target dataset alone, in principle re-centering the features on what is optimal for the target even at the cost of losing generality to other datasets. This step is not required in principle in Typhon, as all Decision Makers (including the one for the target dataset) are kept up to date and viable until the very last epoch. We still include it here to further validate whether such re-centering is still useful for Typhon, and whether allowing the model to focus on dataset-specific features brings a measurable advantage over the standard parallel transfer training.

The hyperparameters for the specialization are necessarily different, especially since Typhon favors particularly small batch sizes, which are typically expected to be sub-optimal in classical training (see Table I for details). The consequences and the overall viability of this approach are presented below in Section V.

Table I: Hyperparameters

	Training phase			**Specialization phase**		
Dataset	**L. rate**	**D. out**	**B. size**	**L. rate**	**D. out**	**B. size**
Prostate	5e-7	0.3	8	8e-10	0.3	64
Brain	5e-7	0.3	8	1e-8	0.3	64
Lung	5e-7	0.3	8	1e-8	0.3	64

Hyperparameters used for the Typhon training phase and subsequent specialization, the latter being an application of classical end-to-end training aimed at further specializing the model to the target dataset. Typhon favors particularly small batch sizes, even though it only trains on one batch from each dataset at each epoch.

IV. Experimental setup

This section presents the setup used to run our experiments. For reproducibility and ease of adoption, we also released our experiments code publicly on GitHub[2], separate from the main Typhon implementation.

A. Hyperparameters

Working with multiple datasets leads to the drawback of having to optimize the hyperparameters of the model multiple times. Table I presents the values we used for our experiments (both training and specialization). The variation is notably minimal across our very different datasets, suggesting that less customization is necessary for Typhon than for other classical applications. Indeed, we found these values empirically by first using the same value for all datasets until obtaining sensible results; from there on, only minimal per-dataset fine-tuning led

[2]Experiments code: https://github.com/eXascaleInfolab/typhon_exp

to the values presented here. This shows that, even though the number of hyperparameters scales with the number of datasets, the work necessary to optimize them **does not**.

B. Reference implementation

Our reference implementation is written in Python, using the PyTorch library for Deep Learning [25]. Specifically, due to parallel transfer, Typhon maintains a dedicated (Adam) optimizer for each Decision Maker, which then sees a single end-to-end network from the corresponding Decision Maker down into the shared Feature Extractor. Our experiments ran on our reference server with a 64-core Intel(R) Xeon(R) 6142 CPU at 2.60GHz, 6GB of RAM per core, and eight NVidia GeForce RTX 2080 Ti GPUs at 2.1GHz with 10GB of GDDR6 vRAM each. To report comparable run times however, each of our experiments was restricted to only utilize a single CPU core and a single GPU.

Still, Typhon turned out to be exceptionally data-efficient, which can be in part linked to favoring a tiny batch size of 8 versus i.e. 128 for Hydra, and often even larger in other literature methods. By disabling performance tracking, which evaluates the model on the training, validation and test set at each epoch to produce our plots, a complete Typhon run (bootstrap, training and specialization) takes **less than three hours** on the single-GPU single-core setup described above, using our three datasets.

C. Typhon model architecture

The Typhon model used in our experiments is once again based on a version of VGG-16 [14] proposed by Song et al. [13] for the classification of cancerous lesions. This choice was selected uniquely for fair comparison with our previous results: exploratory follow-up work using alternative architectures already shows promising results. The Typhon model splits the architecture between feature extraction and decision making at the same point found empirically in our previous work [1], which is past all convolution and pooling layers, plus one fully-connected feed-forward layer. Further details are available in our previous work [1].

D. Data pre-processing

Again for fair comparison with our previous results, the datasets used in our experiments as well as the pre-processing and augmentation methods are identical to our previous work [1]: please refer to our previous publication for complete details. In short, this includes the DWI sequences from the PROSTATEx MRI dataset [9]–[12], with high b-value and restricted to the transversal axis (which is but a fraction of the data available on the PROSTATEx dataset; CT scan from the Lung CT dataset [15]–[17]; and T2W sequences from the Brain Tumor Detection MRI dataset [18]. Data augmentation is also unchanged: a relatively large patch centered on the lesion (for positive cases) is cropped out and then augmented by introducing rotation, flipping and shifting. Class imbalance is also addressed in this step by adjusting the quantity of synthetic data produced for the different classes in each dataset. We picked once again the PROSTATEx dataset as our final target, reusing the exact same partition into train, validation and test sets.

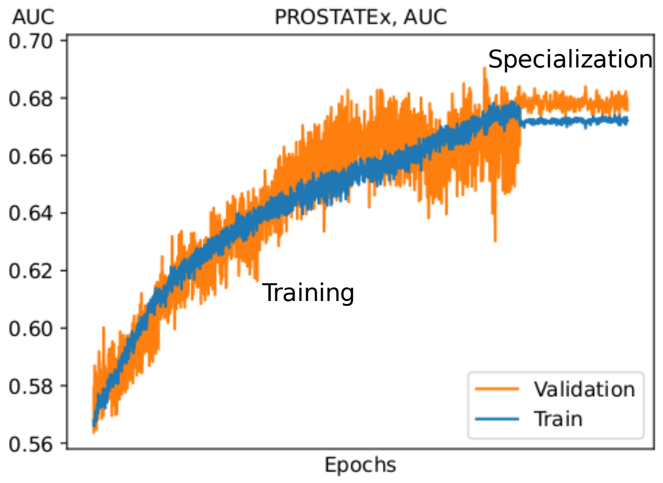

Figure 4: **Training performance.** Plot of model performance computed at each epoch of the training on both the training set and validation set of the PROSTATEx dataset. The sharp discontinuity at the end indicates the transition from the *training phase* to the *specialization phase*, which employs classical end-to-end training on the target dataset alone. Our original hypothesis was that the specialization would learn further dataset-specific features, which are expected to be otherwise lost to catastrophic forgetting when using parallel transfer. These plots show however that classical learning finds little to improve, implying that the training phase is already capable of learning dataset-specific features into the Feature Extractor, rendering the specialization phase unnecessary. The duration of the training and specialization phases are not to scale relatively to each other, only to enhance readability: the training phase ran for 10’000 epochs versus the 300 epochs of specialization, as the training uses a tiny batch size and only sees one batch from each dataset at each epoch. The training is actually faster in wall-clock time than the specialization.

V. Results and discussion

We compare the performance of Typhon directly against our previous Hydra work [1], which includes results from classical end-to-end training and Transfer Learning. Our results are summarized in Table II. Typhon significantly improves on our previous results. Our experiments also highlighted significant improvements in terms of sample efficiency and overall learning ability, as discussed below.

Figure 4 shows the performance of the model during the training and then specialization phases, on both the training and validation sets of our target dataset PROSTATEx, allowing us to make a set of important observations. Firstly, the performance on the PROSTATEx training set increases with the expected logaritmic trend common to most Deep Learning approaches; in our case however this is not to be taken for granted, as our training process is fundamentally different, and a much more noticeable delay was initially expected. At each epoch, Typhon trains the model on one single batch from each of the datasets in turn, not solely on PROSTATEx. This implies that the training objective is varying wildly following the heterogeneity of the datasets, but this only results in minor variations (the "thickness" of the line here). This can be controlled by altering the batch size, and it is what led us to ultimately favor such a small value. This validates our hypothesis that **the moving target problem does not hinder our parallel transfer training process**, even when observed from a single-dataset perspective, and can thus be leveraged for our intended purposes (which is, as a generalization focus) without loss in performance.

The second point to notice in Figure 4 is that the model performance on the validation set follows the performance on the training set for the entire duration of our training. This is another unexpected observation, as most methods see the validation curve peak relatively early, then deflate as the model progressively overfits on the data. This can be seen clearly in Figure 5 (on the PROSTATEx test set this time), put in perspective by the performance of Transfer Learning and, to a lesser extent, also of Hydra.

Overfitting can critically impair Deep Learning methods: if the model is sufficiently complex w.r.t. the available data, learning by heart can be simpler and thus preferred over the complex processing of increasingly more abstract features, reducing its performance on unseen data (generalization). In the case of Typhon however, overfitting is staved off if not entirely combated: the model keeps performing just as well on the validation set as it does on the training set, even at the peak of its performance. This has two major implications: (i) we can expect the test results to also follow the same trend of constant improvement as the training progresses; and most importantly (ii) **Typhon shows significant potential to address overfitting**, even in tasks not hindered by data scarcity. Delaying overfitting means that the training can continue for longer, and more information can be absorbed from the data, without having to stop the process for fear of the model losing its generalization capability.

One final deduction from Figure 4 regards the transition from the training phase to the specialization phase, which is starkly noticeable in the plot. The specialization is implemented as classical end-to-end training on a single, target dataset, which can be seen as "initialized" using Typhon's parallel transfer training on multiple datasets. As expected, the variance in AUC is greatly reduced for both curves, as focusing on a single dataset entirely rids of the moving target problem. At the same time however, the performance improvement is minimal, within 1%, which is insignificant given the performance fluctuation during Typhon training and reductive with respect to the jump seen in Hydra's specialization performance (Figure 5; on the test set). This means that **classical learning struggles to improve on a model already learned by Typhon**, implying that Typhon eventually learns even dataset-specific information, to the point that no further improvement

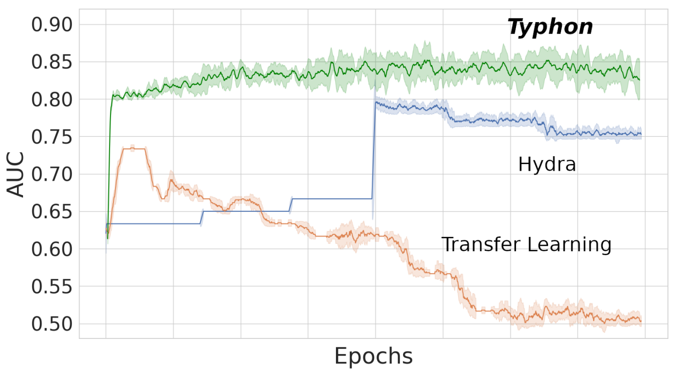

Figure 5: **Performance comparison.** Evolution of model performance (AUC) on the target PROSTATEx test set, as the training progresses. While the results on the validation sets are biased, as they are used for model selection, the test set is never accessed by the training and thus represents a better proving ground for the model's generalization and applicability. The results of Typhon are not only better in terms of pure AUC, but also in terms of the number of samples seen, since the epochs for the three curves are not represented at the same scale. To compare the number of samples seen, refer to Table II. Typhon also shows remarkable consistence in its performance, possibly due to its custom bootstrapping technique: the line is built from an average over 5 runs, with variance shown in lighter color. The boost in performance w.r.t. the validation AUC numbers is due to the validation set including synthetically augmented data, making it a more challenging task than the original data alone as used in the test set.

is possible on the available data. The Typhon learning phase was fundamentally still able to "overfit" the feature extractor to dataset-specific features, without hindering the generalization across the other datasets, at least to the level of making the specialization phase superfluous. This is against our original hypothesis that only generic features can survive Typhon's combination of moving target and catastrophic forgetting, and further strengthens its results beyond our original expectations.

This brings us to Figure 5, which depicts the performance of the model on the PROSTATEx test set as the training progresses, and compares Typhon with Hydra and Transfer Learning – and thus implicitly also with classical learning, as the target dataset is the first target of the Transfer Learning process (the first clean rise in its plot). As we aim to compare each method's *capability* and potential, beyond the specific performance on our arbitrary selection of datasets, the horizontal axis labeled "Epochs" is not re-scaled (i.e., the number of epoch varies depending on the method). Typhon uses a much smaller batch size (see Table I), and only trains on one batch from each dataset at each epoch, which results in a proportionally higher number of epochs, but a quicker wall-clock training time.

Here we see Typhon jumping ahead in performance from a very early stage, and remaining ahead while consistently

Table II: AUC on PROSTATEx validation and test set

Dataset	Classical	Transfer	Hydra	Typhon
Samples seen	50M	152M	356M	78M
Validation	0.73	0.86	0.77	0.68
Test	0.68	0.72	0.80	**0.87**

Columns correspond to classical end-to-end learning (using only the target dataset), Transfer Learning (using all datasets in turn, the target being used twice, as first and then again as last), Hydra, and our new contribution Typhon. Training Typhon on three datasets uses only 56% more samples than training a classical Deep Learning model on just one dataset, and 78% fewer samples than what Hydra required on three datasets.

improving albeit at a slower rate. As these results are on the test data, which is of course inaccessible to the training: the continuous improvement defies our reasonable expectation of overfitting slowly but surely setting in. This validates our initial interpretation of Figure 4, i.e. that Typhon's focus on generalization actively combats and delays overfitting, even while obtaining the highest results on the target dataset. A clearer estimation of Typhon's ability in this sense is left for future work, with exploratory work already looking promising.

VI. Conclusions

This paper introduces Typhon, a Deep Learning framework to train a single model using multiple heterogeneous datasets leveraging parallel transfer, focused on generalization to the point of actively combating and delaying overfitting.

Parallel transfer implies training in turn on each dataset at every epoch, which is challenging to integrate into a learning method because of the compounding problems of *moving target* and *catastrophic forgetting*. Any information learned could be quickly overwritten as the learning target constantly jumps between datasets, unless it generalizes well across all of them.

Typhon leverages this as a feature rather than a limitation, by introducing a special architecture which explicitly partitions the network layers based on their expected role. The first layers, typically committed to constructing increasingly complex features (such as convolution and pooling) constitute the Feature Extractor, while the last layers, commonly dedicated to decision making (typically fully connected) constitute the Decision Maker. The latter are then duplicated to match one dedicated Decision Maker with each dataset. The Feature Extractor is then trained using data from all datasets, with moving target and catastrophic forgetting allowing only for generic features to be retained. Meanwhile the dedicated Decision Makers specialize as far as its smaller architecture allows, creating full end-to-end models when paired with the common Feature Extractor.

Designing a training method under such assumptions led to a set of fundamental challenges rarely encountered outside parallel transfer, such as dataset imbalance and training divergence following moving targets, particularly on randomly initialized models. The core of this work discusses our effort and results in overcoming those problems, ultimately making the method viable. We kept our experimental setup constant for fair comparison, including the neural network architecture, datasets, and data pre-processing, and present here our Typhon results against classical learning, transfer learning and Hydra.

Typhon shows a 7% increase in AUC over Hydra, which compounds to 15% over classical learning and 12% over transfer learning, while showing higher sample efficiency and a notable resilience to overfitting. We released our reference implementation open source, paired with our experimental setup on a dedicated repository, for reproducibility and ease of adoption.

A. Broader impact

Medical imaging applications, particularly in support to radiologists in clinical cancer detection applications, has a direct impact to cancer casualties. Expert radiologists are a limited and highly-valuable resource, and while early detection correlates to higher survivability rates, humans performance is also negatively impacted by long working hours and heightened stress levels. Computer-aided diagnostic tools are not troubled by the long working hours expected of clinical staff, making them potentially invaluable in supporting the daily routine of a Radiology department.

Over the past years, the attempts to bridge the gap between the performance of Deep Learning methods for visual classification and medical applications have been constantly held back by the problem of data scarcity. This is of course linked to patient privacy: creating large datasets of high-quality medical imaging requires not only considerable effort, but specific written authorizations from each patient, validation from the local ethical committee, as well as significant contribution from expert radiologists required to label the collected data. Furthermore, the resulting data is always subject to data quality issues, due to the lack of an objective “golden standard”, and the inherent subjectivity of human diagnosis.

High-performance Computer Aided Diagnostic (CAD) tools are among the most impactful tools available in our constant battle with cancer at the diagnosis stage. We believe that our new approach, Typhon, allows to side-step the data scarcity problem that affect these applications, opening the path for significant advances in Deep Learning applications in this context.

B. Future work

Creating larger and larger datasets in applications where obtaining data is often costly or restricted (such as medical imaging) has proven to be a fundamentally insurmountable challenge. Year over year however, the number of available datasets grows constantly. Typhon is designed to leverage this trend, by allowing utilizing multiple, diverse datasets while guaranteeing a global, unified improvement in model performance. As we work side by side with clinical radiologists in our team, we are looking forward to put forward and test a production-ready implementation of our framework as a support tool for radiologists.

Future work will also include customizing the network architecture to suit specific applications, the introduction of newer and larger datasets (including proprietary ones), and moving beyond classification and into localization and segmentation tasks. Finally, the full extent of Typhon's ability to stave off overfitting in very deep models requires further investigation, as the potential applications for the whole field of Deep Learning could be significant even in tasks where data scarcity is not an issue.

Acknowledgments

This work received support from the Swiss National Science Foundation under grant number 320038_176229/1.

References

[1] G. Cuccu, J. Jobin, J. Clément, A. Bhardwaj, C. Reischauer, H. Thöny, and P. Cudré-Mauroux, "Hydra: Cancer detection leveraging multiple heads and heterogeneous datasets," in *2020 IEEE International Conference on Big Data, BigData*, 2020.

[2] M. T. Rosenstein, Z. Marx, L. P. Kaelbling, and T. G. Dietterich, "To transfer or not to transfer," in *NIPS 2005 workshop on transfer learning*, vol. 898, 2005, pp. 1–4.

[3] L. Torrey and J. Shavlik, "Transfer Learning," *Handbook of Research on Machine Learning Applications and Trends: Algorithms, Methods, and Techniques*, 2010.

[4] S. J. Pan and Q. Yang, "A survey on transfer learning," *IEEE Transactions on knowledge and data engineering*, vol. 22, no. 10, pp. 1345–1359, 2009.

[5] A. Abubakar, M. Ajuji, and I. Yahya, "Comparison of deep transfer learning techniques in human skin burns discrimination," *Applied System Innovation*, vol. 3, p. 20, 04 2020.

[6] R. K. Samala, H.-P. Chan, L. M. Hadjiiski, M. A. Helvie, K. H. Cha, and C. D. Richter, "Multi-task transfer learning deep convolutional neural network: application to computer-aided diagnosis of breast cancer on mammograms," *Physics in Medicine & Biology*, vol. 62, no. 23, pp. 8894–8908, nov 2017.

[7] W. Zhang, R. Li, T. Zeng, Q. Sun, S. Kumar, J. Ye, and S. Ji, "Deep model based transfer and multi-task learning for biological image analysis," *IEEE Transactions on Big Data*, vol. 6, no. 2, pp. 322–333, 2020.

[8] R. Caruana, "Multitask learning," *Machine learning*, vol. 28, no. 1, pp. 41–75, 1997.

[9] G. Litjens, O. Debats, J. Barentsz, N. Karssemeijer, and H. Huisman, "ProstateX Challenge data," *The cancer imaging archive*, 2017. [Online]. Available: https://wiki.cancerimagingarchive.net/display/Public/SPIE-AAPM-NCI+PROSTATEx+Challenges

[10] ——, "Computer-aided detection of prostate cancer in MRI," *IEEE Transactions on Medical Imaging*, vol. 33, no. 5, pp. 1083–1092, May 2014.

[11] K. Clark, B. Vendt, K. Smith, J. Freymann, J. Kirby, P. Koppel, S. Moore, S. Phillips, D. Maffitt, M. Pringle, L. Tarbox, and F. Prior, "The Cancer Imaging Archive (TCIA): Maintaining and operating a public information repository," *Journal of Digital Imaging*, vol. 26, no. 6, pp. 1045–1057, Dec. 2013.

[12] S. G. Armato, H. Huisman, K. Drukker, L. Hadjiiski, J. S. Kirby, N. Petrick, G. Redmond, M. L. Giger, K. Cha, A. Mamonov, J. Kalpathy-Cramer, and K. Farahani, "PROSTATEx Challenges for computerized classification of prostate lesions from multiparametric magnetic resonance images," *Journal of Medical Imaging*, vol. 5, no. 04, p. 1, Nov. 2018.

[13] Y. Song, Y.-D. Zhang, X. Yan, H. Liu, M. Zhou, B. Hu, and G. Yang, "Computer-aided diagnosis of prostate cancer using a deep convolutional neural network from multiparametric MRI: PCa classification using CNN from mp-MRI," *Journal of Magnetic Resonance Imaging*, vol. 48, no. 6, pp. 1570–1577, Dec. 2018.

[14] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv:1409.1556 [cs]*, Apr. 2015.

[15] S. G. Armato, K. Drukker, F. Li, L. Hadjiiski, G. D. Tourassi, R. M. Engelmann, M. L. Giger, G. Redmond, K. Farahani, J. S. Kirby, and L. P. Clarke, "LUNGx Challenge for computerized lung nodule classification," *Journal of Medical Imaging*, vol. 3, no. 4, p. 044506, Dec. 2016.

[16] S. G. Armato, L. Hadjiiski, G. D. Tourassi, K. Drukker, M. L. Giger, F. Li, G. Redmond, K. Farahani, J. S. Kirby, and L. P. Clarke, "Guest Editorial: LUNGx Challenge for computerized lung nodule classification: reflections and lessons learned," *Journal of Medical Imaging*, vol. 2, no. 2, p. 020103, Jun. 2015.

[17] S. G. Armato, L. Hadjiiski, G. D. Tourassi, M. L. Giger, F. Li, G. Redmond, K. Farahani, J. Kirby, and L. P. Clarke, "SPIE-AAPM-NCI Lung Nodule Classification Challenge Dataset," *The cancer Imaging Archive*, 2017. [Online]. Available: https://wiki.cancerimagingarchive.net/display/Public/SPIE-AAPM+Lung+CT+Challenge#b38bc90c1f4c498fbcb2acb3495cd9d8

[18] "Brain MRI images for brain tumor detection," 2019. [Online]. Available: https://kaggle.com/navoneel/brain-mri-images-for-brain-tumor-detection

[19] J. Schmidhuber, S. Hochreiter, and Y. Bengio, "Evaluating benchmark problems by random guessing," *A Field Guide to Dynamical Recurrent Networks, ed. J. Kolen and S. Cremer*, pp. 231–235, 2001.

[20] A. Gosain and S. Sardana, "Handling class imbalance problem using oversampling techniques: A review," in *2017 international conference on advances in computing, communications and informatics (ICACCI)*. IEEE, 2017, pp. 79–85.

[21] G. E. Batista, R. C. Prati, and M. C. Monard, "A study of the behavior of several methods for balancing machine learning training data," *ACM SIGKDD explorations newsletter*, vol. 6, no. 1, pp. 20–29, 2004.

[22] N. V. Chawla, N. Japkowicz, and A. Kotcz, "Special issue on learning from imbalanced data sets," *ACM SIGKDD explorations newsletter*, vol. 6, no. 1, pp. 1–6, 2004.

[23] X.-Y. Liu, J. Wu, and Z.-H. Zhou, "Exploratory undersampling for class-imbalance learning," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 39, no. 2, pp. 539–550, 2008.

[24] G. Cuccu, S. Danafar, P. Cudré-Mauroux, M. Gassner, S. Bernero, and K. Kryszczuk, "A data-driven approach to predict nox-emissions of gas turbines," in *2017 IEEE International Conference on Big Data, BigData 2017, Boston, MA, USA, December 11-14, 2017*, 2017, pp. 1283–1288. [Online]. Available: https://exascale.info/assets/pdf/cuccu2017bigdata.pdf

[25] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 2019, pp. 8024–8035. [Online]. Available: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf