HAL
open science

Deep multi-task mining Calabi-Yau four-folds

Harold Erbin, Riccardo Finotello, Robin Schneider, Mohamed Tamaazousti

To cite this version:

Harold Erbin, Riccardo Finotello, Robin Schneider, Mohamed Tamaazousti. Deep multi-task mining Calabi-Yau four-folds. Mach.Learn.Sci.Tech., 2022, 3 (1), pp.015006. 10.1088/2632-2153/ac37f7 . hal-03323024

HAL Id: hal-03323024

https://hal.science/hal-03323024v1

Submitted on 26 Apr 2023

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

MIT-CTP/5319
UUITP-36/21

Deep multi-task mining Calabi-Yau four-folds

Harold Erbin*[1,2,3], Riccardo Finotello†[3,4], Robin Schneider‡[5], and Mohamed Tamaazousti§[3]

[1] *Center for Theoretical Physics, Massachusetts Institute of Technology*
Cambridge, MA 02139, USA
[2] *NSF AI Institute for Artificial Intelligence and Fundamental Interactions*
[3] *Université Paris Saclay, CEA, LIST*
Palaiseau, F-91120, France
[4] *Université Paris Saclay, CEA, Service d'Études Analytiques et de Réactivité des Surfaces (SEARS)*
Gif-sur-Yvette, F-91191, France
[5] *Department of Physics and Astronomy, Uppsala University*
SE-751 20 Uppsala, Sweden

November 16, 2021

arXiv:2108.02221v2 [hep-th] 15 Nov 2021

Abstract

We continue earlier efforts in computing the dimensions of tangent space cohomologies of Calabi-Yau manifolds using deep learning. In this paper, we consider the dataset of all Calabi-Yau four-folds constructed as complete intersections in products of projective spaces. Employing neural networks inspired by state-of-the-art computer vision architectures, we improve earlier benchmarks and demonstrate that all four non-trivial Hodge numbers can be learned at the same time using a multi-task architecture. With 30 % (80 %) training ratio, we reach an accuracy of 100 % for $h^{(1,1)}$ and 97 % for $h^{(2,1)}$ (100 % for both), 81 % (96 %) for $h^{(3,1)}$, and 49 % (83 %) for $h^{(2,2)}$. Assuming that the Euler number is known, as it is easy to compute, and taking into account the linear constraint arising from index computations, we get 100 % total accuracy.

Contents

1 Introduction 1

2 Related works 3

3 Exploring the dataset 3
3.1 CICY four-folds 3
3.2 Classifying Hodge numbers 4

4 CICYMiner 7
4.1 Preprocessing and Evaluation Strategy 8
4.2 Training 9
4.3 Results 9
4.4 Ablation study 9

5 Conclusion 11

References 13

*erbin@mit.edu
†riccardo.finotello@cea.fr
‡robin.schneider@physics.uu.se
§mohamed.tamaazousti@cea.fr

1 Introduction

There is a growing body of research that applies modern techniques from data science to problems in string theory [1]. The reasons for that are two-fold. On the one hand, standard computations in string theory are hard, in particular they can be NP-hard or even undecidable [1–3]. Due to double exponential scaling laws in terms of computational resources with respect to the input parameters, string theory calculations often fail to finish in a reasonable amount of time even on modern machines. On the other hand, there are too many configurations to consider. The largest estimates put a bound of $\mathcal{O}(10^{272\,000})$ when considering F-theory compactified on a Calabi-Yau four-fold [4]. Parsing that many configurations

is impossible, thus computational smart ways are needed to select potentially interesting vacuum configurations [5, 6].

An important key component in realistic string theory compactifications are Calabi-Yau manifolds. These manifolds have been studied extensively in the past, and thus they comprise some of the best datasets within the string theory community [7]:

1. The first widely used dataset are the 7890 complete intersection Calabi-Yau, in short CICY, manifolds in three complex dimensions by Candelas et al. [8–10].

2. The largest dataset, the Kreuzer-Skarke list, contains 473 million reflexive polytopes in four dimensions. These encode a toric ambient space, from which one obtains Calabi-Yau three-folds by considering the hypersurface defined by the canonical bundle [11].

3. CICY four-folds have also been classified and amount to 921 497 distinct configuration matrices [12, 13].

The incredible progress in data science, in particular image recognition, over the past decade can in part be attributed to large and clean datasets [14]. They allowed researchers to benchmark their algorithms and let the best ones compete against each other, which in turn resulted in rapid development and ever improving neural network architectures [15–19]. We will proceed in a similar vein in this paper. The number of independent Kähler moduli of CICY three-folds has been successfully analyzed using neural networks in the past. These benchmarks were initiated by He, who proposed to treat their configuration matrices as a simple two-dimensional image [20].

In previous works [21, 22], two of the authors have shown that learning $h^{(1,1)}$ is possible to great accuracy, but the limited training data is not sufficient to generalize the learning to the number of complex structure moduli $h^{(2,1)}$. Computing Hodge numbers of Calabi-Yau manifolds is of great importance, since cohomology computations are an integral part of string theory compactifications. They for example determine the number of massless fermion generations in string theory compactifications. Thus, the goal is to identify performant algorithms in these well-studied datasets of tangent bundle cohomologies, which can then generalize to more complicated vector bundles.

In the rest of this paper, we will present two different approaches for learning Hodge numbers of CICY four-folds. First, we will treat the problem as a standard image classification task where the Hodge numbers are the image labels. For this purpose, we employ an Inception module based architecture [16–18, 21, 22] and show that a single set of hyperparameters generalizes well to all four Hodge numbers, yielding a mean accuracy over all Hodge numbers of 85 %. This suggests that we could scale the approach to a multi-task learning problem.

Subsequently, we show that all Hodge numbers can be learned simultaneously by utilizing a branched network with hard parameter sharing [23, 24] between the task specific substructures, which ultimately are responsible for learning the distributions of the Hodge numbers. The multi-task approach has several advantages, with respect to single-task architectures. From a technical side, multi-task learning has been shown to improve the overall performance of the models [23]. From a physics and algebraic geometry perspective, a single model hints towards the definition of a unified framework from which it may be possible to extract meaningful theoretical information, such as closed form formulas. The model we developed is capable of learning at the same time, and without rescaling, the four dimensions of the tangent space cohomologies of CICYs, accounting for the heavy class imbalance present in the dataset. This multi-task Ansatz leads to perfect performance on two of the four Hodge numbers and accuracy of 96 % and 83 % for $h^{(3,1)}$ and $h^{(2,2)}$ respectively, with a training ratio of 80 %.

The outline of this paper is as follows. In Section 2, we discuss related works of learning cohomologies and earlier results on Calabi-Yau three-folds. Section 3 explores the dataset of CICY four-folds and presents the results of our classification experiments. This is followed by our main results in Section 4 in which we introduce our deep learning model *CICYMiner*, a multi-task regression model based on chained Inception modules that predicts all four Hodge numbers at once. We conclude in Section 5 with some outlooks. Python codes for this paper can be found at:

- https://github.com/robin-schneider/cicy-fourfolds
- https://github.com/thesfinox/ml-cicy-4folds

The list of packages used throughout the development comprises `pandas` [25, 26] and `numpy` [27] for data operations, `matplotlib` [28] and `seaborn` [29] for visualisation, and `tensorflow` [30] for the deep learning algorithms.

2 Related works

The first paper utilizing machine learning algorithm to predict various different cohomology dimensions was written by He [20]. The author tackled the problem of predicting Hodge numbers of CICY three- and four-folds, but also line bundles over these manifolds [20]. These studies have later been extended to systematically investigate CICY three-folds with linear regression, support vector machines, and dense neural networks achieving accuracies ranging from 37 % to 85 % [31, 32] when using 70 % training data. The benchmarks have subsequently been improved by using an Inception-based architecture to accurately predict 97 % of the test data using only 30 % training data, essentially solving the problem of predicting $h^{(1,1)}$ [21, 22]. This work was supplemented by more methodological studies in which the dataset was augmented with various other (topological) quantities. Other works on CICY three-folds include [33, 34].

An initial exploration of CICY four-folds has been started by He and Lukas [20, 35]. The authors used a simple dense neural network and were able to predict $h^{(1,1)}$ with an accuracy of 96 %. This promising early result showed that the increased size of the dataset improves the performance significantly. However, in line with previous studies of $h^{(2,1)}$ on CICY three-folds, the authors were unable to accurately predict the value of the other Hodge numbers, reaching an accuracy of only 27 % for $h^{(3,1)}$. They were successful in improving this accuracy for a subset of the dataset by considering all configuration matrices of shape $(4, 4)$ and using feature enhancement. This feat was achieved by supplementing the training samples with all up to degree four monomials of the defining polynomials and pushed the accuracy to 95 %.

The Kreuzer-Skarke list has also been the target of deep learning algorithms. In order to identify equivalent Calabi-Yau manifolds coming from different triangulations, Demirtas et al. trained residual neural networks to learn the triple intersection numbers [36]. They reached an almost perfect performance, which allowed them to cut down the computation time from seconds to microseconds. This in turn made it possible to derive an upper bound on the number of distinct Calabi-Yau manifolds arising from the polytope with the most triangulations, setting it to 10^{428}.

There are several ongoing projects in learning Hodge numbers of line bundle cohomologies. These can be separated into two different approaches. First, learning the cohomology dimensions directly, for example on del Pezzo surfaces [37] and on CICY three-folds [6, 20, 38, 39]. Second, neural networks have been used to classify cones in the cohomology-dimension landscape [40–42]. The Hodge numbers belonging to these cones can all be described by the same analytic equations [43].

3 Exploring the dataset

In this section, we will introduce and explore complete intersection Calabi-Yau four-folds. We then proceed to learn the four non-trivial Hodge numbers independently using neural networks with an Inception inspired architecture [16–18].

3.1 CICY four-folds

A complete intersection Calabi-Yau manifold is fully defined by its configuration matrix. This matrix encodes the polynomial degrees and ambient space factors in the following way:

$$\mathcal{M} = \left[\begin{array}{c||ccc} n_0 & p_1^0 & \cdots & p_K^0 \\ \vdots & \vdots & \ddots & \vdots \\ n_r & p_1^r & \cdots & p_K^r \end{array}\right]_\chi . \tag{3.1}$$

Each $p_j^i \in \mathbb{N}$ is the degree of the j-th polynomial in the homogeneous coordinates of the i-th complex projective space with dimension n_i. The Calabi-Yau condition is translated in the configuration matrix by requiring that

$$n_i + 1 = \sum_{j=1}^{K} p_j^i . \tag{3.2}$$

The Euler number χ is given in the subscript and can be directly computed by integrating the fourth Chern class or from the four non-trivial Hodge numbers as

$$\chi = 4 + 2h^{(1,1)} - 4h^{(2,1)} + 2h^{(3,1)} + h^{(2,2)} . \tag{3.3}$$

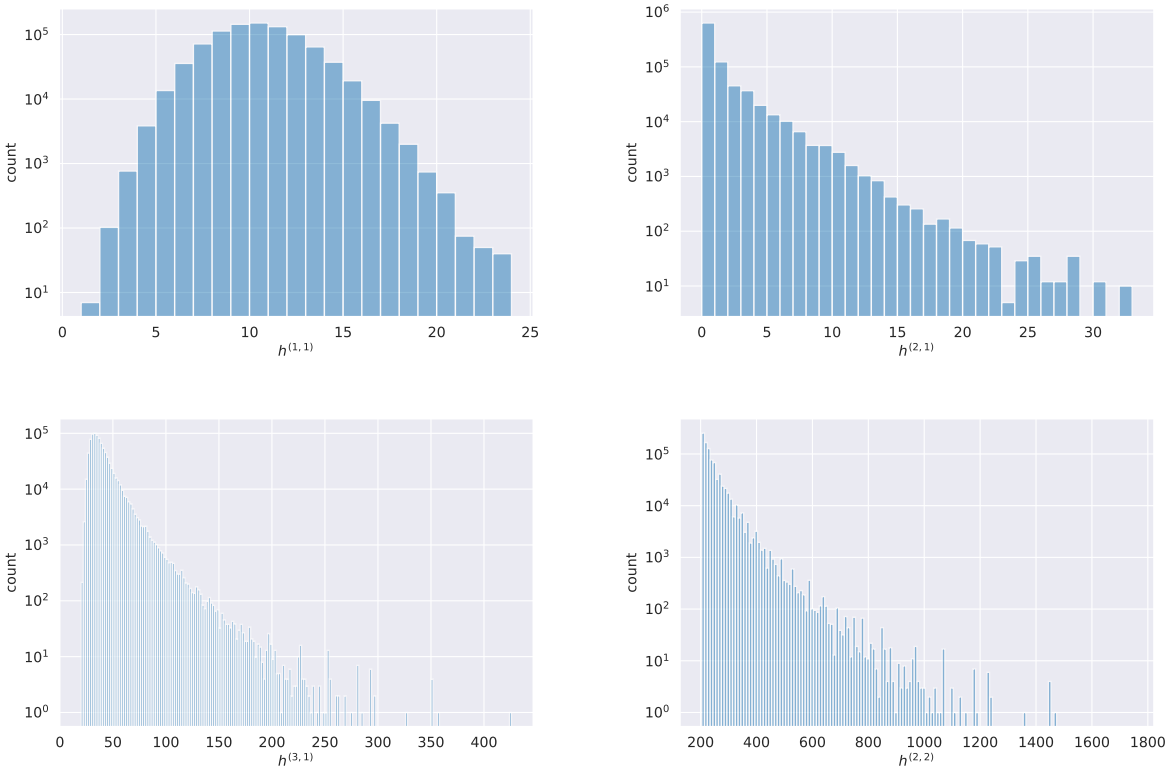


Figure 1: *The plots show the histograms with logarithmic y-axis of the four non-trivial Hodge numbers. In the first row we have on the left the distribution of $h^{(1,1)}$, to the right of $h^{(2,1)}$. In the bottom row $h^{(3,1)}$ is presented to the left and $h^{(2,2)}$ to the right.*

A second linear relationship between the Hodge numbers can be derived by combining the indices $\chi_q = \chi(\mathcal{M}, \wedge^q T\mathcal{M}^*)$ [13] leading to

$$44 = -4h^{(1,1)} + 2h^{(2,1)} - 4h^{(3,1)} + h^{(2,2)}. \quad (3.4)$$

The configuration matrices have been generated from an initial set of matrices and subsequently applying the splitting procedure [8, 12], finding new manifolds and discarding equivalent descriptions. In this way, a total of 921 497 topological distinct types of CICY manifolds were found, with 905 684 of them not being direct products of lower dimensional manifolds.

The Hodge number distributions are presented in Figure 1. The mean, maximum and minimum values are

$$\langle h^{(1,1)} \rangle = 10.1^{24}_{1}, \quad \langle h^{(2,1)} \rangle = 0.817^{33}_{0},$$
$$\langle h^{(3,1)} \rangle = 39.6^{426}_{20}, \quad \langle h^{(2,2)} \rangle = 241^{1752}_{204}. \quad (3.5)$$

Notice that the distributions of the Hodge numbers are, in general, imbalanced: for instance, $h^{(2,1)}$ vanishes for 70 % of the configuration matrices in the dataset. We find that 54.5 % are favourable (i.e. $h^{(1,1)}$ is equal to the number of projective spaces), less than the 61.9 % for CICY three-folds.[1] Hence, for slightly more than half of the cases we have $h^{(1,1)} = r$, the number of projective ambient space factors. This number is important as it should be the baseline to compare any algorithm against.

3.2 Classifying Hodge numbers

Problems in image recognition are usually formulated as classification tasks. Take the *ImageNet* dataset which consists of 14×10^6 data points with over 21 000 classes. That is about one order of magnitude larger, both in samples and classes, than predicting $h^{(2,2)}$. In this section, we will train one neural network to classify each of the four non-trivial Hodge numbers independently. We will use an architecture based on Inception modules [16–18] as was done for the best performing predictors of the CICY three-fold Hodge numbers [21, 22]. This specific architecture has been shown to lead to the best performance on the

[1] There exists another dataset of CICY three-folds in which 99.1 % are favourable [10], but no such feature enhanced data is available for the four-folds. However, the results from [21, 22] show that using favourable matrices helps mostly in computing $h^{(1,1)}$.

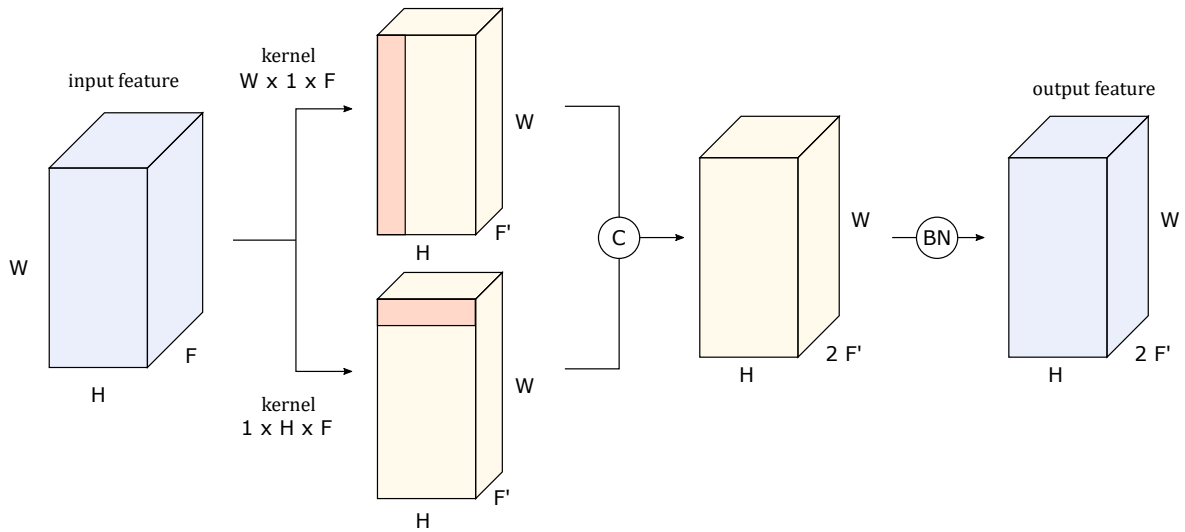


Figure 2: *An Inception module can be decomposed into the different convolutional kernels scanning over the width (W) and height (H) with filters (F). They are subsequently concatenated (C) and followed by a batch normalization (BN) layer. The Inception module is the main building block of both the CICYMiner and the classification architectures.*

configuration matrices when using 1d kernels of maximal size. This partially reflects the fact that scanning coordinates in each projective space and a single variable over all projective spaces helps in better learning the connections between the different hypersurfaces of the CICYs (see Figure 2). The choice of maximal 1d kernel is, in fact, motivated by the mathematical machinery required to compute Hodge numbers. There, one has to compute the dimension of ambient space cohomology group representations, which are stacked for each projective space. These ambient space representations arise after splitting up the Koszul resolution

$$0 \to \wedge^K \mathcal{N}^* \to \ldots \to \mathcal{N}^* \to \mathcal{O}_{\mathcal{A}} \to \mathcal{O}_{\mathcal{A}}|_{\mathcal{M}} \to 0 \,. \tag{3.6}$$

which contains the antisymmetric products $\wedge^s \mathcal{N}^*$ of the defining hypersurfaces ($\mathcal{N}$ denotes the normal bundle, which contains the information about the polynomial degrees p^i_j). Moreover, using an Inception based architecture lead to a performance increase of misclassification rate on the *ImageNet* dataset from 15.3 % for AlexNet [15] using a standard convolutional architecture to 6.7 % for the first version of GoogLeNet [16].

We proceed as in earlier studies [21, 22] by considering different train:val:test splits with respectively 10 %, 30 %, 50 % and 80 % training and 10 % validation data. The architecture hyperparameters have been optimized using Bayesian Optimization Hyperband [44, 45] on the problem of predicting $h^{(3,1)}$. The same hyperparameters have then been used to also classify the other three Hodge numbers.

We opted to present the results of neural networks with a comparable number parameters $840\,000 \pm 10\,000$ to the number of configuration matrices. This architecture comprises four Inception modules, with respectively 3×64 and 16 filters, utilizing batch normalization for better gradient propagation into the earlier layers [16–18, 46]. Figure 2 decomposes an Inception module into its different ingredients. The convolutional kernels scan over the configuration matrix dimensions, i.e. the maximal number of possible projective ambient spaces (16) and the maximal number of polynomial constraints (20). The Inception modules are followed by three dense layers with 16 units, ReLU activation function and dropout layers with a 0.2 rate to contrast overfitting. Furthermore, we employ ℓ_1 (10^{-5}) and ℓ_2 (10^{-6}) regularization for all weights in the network. The last layer contains a softmax activation function with $\{h^{(i,j)}_{\min}, \ldots, h^{(i,j)}_{\max}\}$ classes. The network is trained with Adam optimizer and an initial learning rate of 4×10^{-4} on a 32 minibatch size. This architecture is still trainable in a reasonable amount of time on a desktop computer with access to a GPU. In comparison to earlier studies [21, 22], we found that leaving the outliers inside the training data does not negatively impact the results.

Figure 3 shows in the top row the training loss and accuracy, and in the bottom row validation accuracy tracked over the training process and

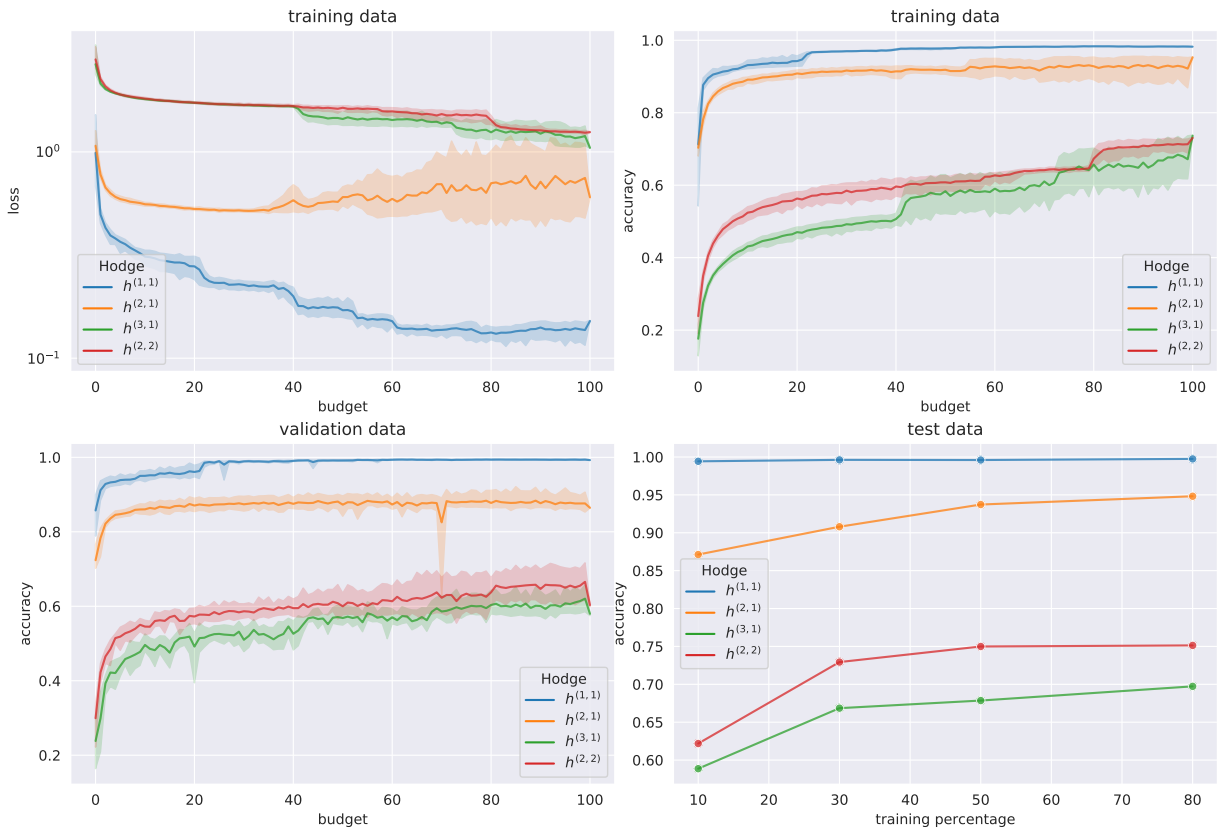


Figure 3: *The first row shows training loss and accuracy plotted against the computation budget. The error bars represent the upper and lower bounds for the four different training ratios. In the second row, we plot on the left the validation accuracy and on the right the test accuracy of the best performing models for the different training ratios.*

test accuracy for the best-performing model. The best-performing model is the one with the highest validation accuracy, which one would get when employing early stopping on that metric. It is important to track the best performing models as sometimes the loss starts increasing again as visible from the $h^{(2,1)}$ curve. The error bars are computed from the different training ratios and the budget on the x-axis is given by

$$\text{budget} = \text{number of epochs} \times \frac{\text{percentage of training data}}{80}. \quad (3.7)$$

We observe that $h^{(1,1)}$ is predicted with almost perfect accuracy for any training ratio, while the accuracies of the other three Hodge numbers improve with more training data. However, when the training data contains more than 30 % of the samples one has diminishing returns for the accuracy. This is in line with previous observations for the CICY three-folds [21, 22]. Even though the hyperparameters have been optimized to learn $h^{(3,1)}$, it is the worst performing value. This is interesting as Figure 1 shows that the distribution of $h^{(2,2)}$ spans a longer range, contains more outliers and has a thicker tail. The plots show that we avoid overfitting to the training data.

	$h^{(1,1)}$	$h^{(2,1)}$	$h^{(3,1)}$	$h^{(2,2)}$
10 %	0.99	0.87	0.59	0.62
30 %	**1.00**	0.91	0.67	0.73
50 %	**1.00**	0.94	0.68	**0.75**
80 %	**1.00**	**0.95**	**0.70**	**0.75**
mean	1.00	0.92	0.66	0.71

Table 1: *Comparison of the test accuracy for different training ratios.*

Table 1 collects the accuracy at different training ratios and the mean-value for the four different training ratios of the best performing model.[2] In the training process, we employed learning rate decay with a factor of 0.4, when the validation ac-

[2] Using a five-fold increase in network weights (4×10^6) one is able to improve the accuracy of $h^{(3,1)}$ and $h^{(2,2)}$ to over 80 %. However, this comes at the cost of significant more training time and we then enter the regime where there are more weights than samples in the dataset.

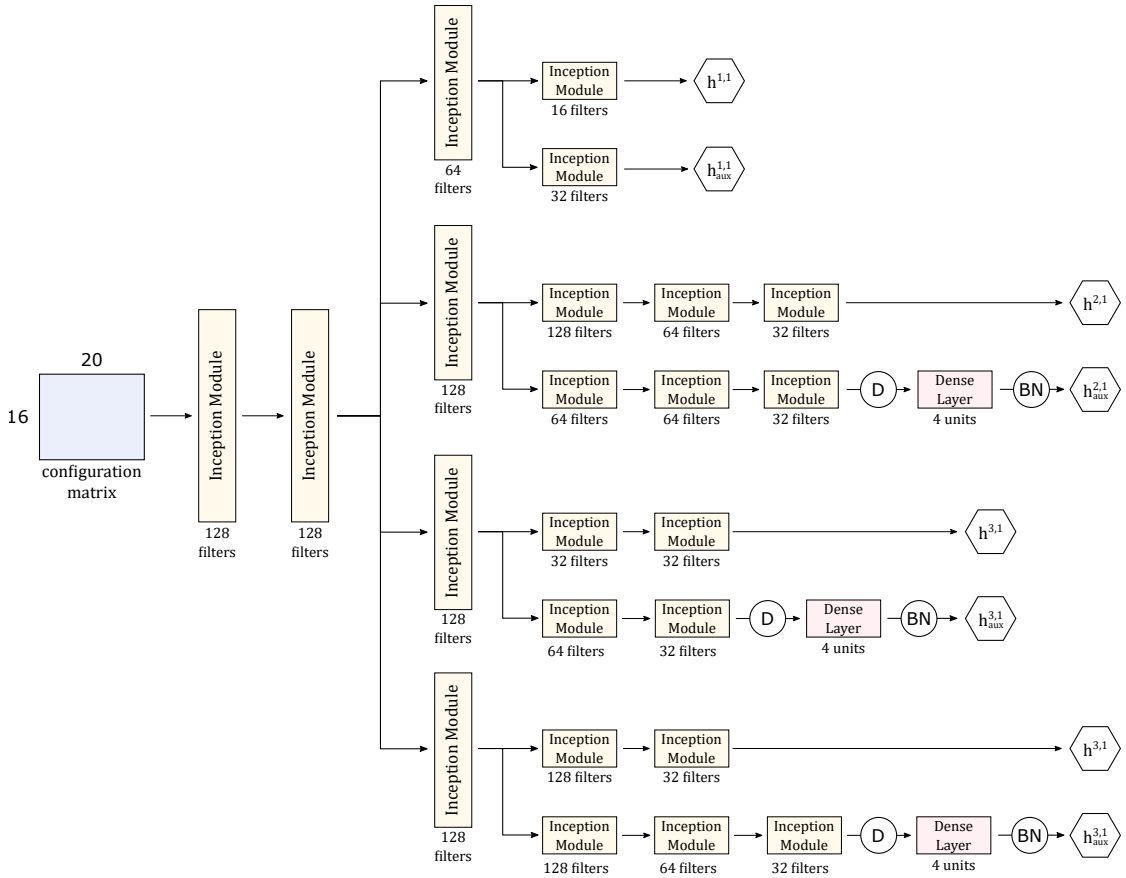


Figure 4: *The basic building block of CICYMiner are Inception modules. The architecture is built to enable the hard parameter sharing in the bottom layer, in order to construct a common representation of the input. The task specific sub-structures then replicate the behaviour through an auxiliary branch, which further uses dense layers, batch normalization (BN) and dropout (D) to control overfitting. The final model predicts all Hodge numbers at once. The composition of the Inception modules are shown in Figure 2.*

curacy did not improve for epochs equivalent to 0.15 × budget. This is clearly visible from the loss and accuracy plots in the top row and accounts for the down- and up-stairs steps. Summarizing the results, we find that the hyperparameters found for predicting the worst performing Hodge number $h^{(3,1)}$ also generalize well to the other three Hodge numbers. This is a first indication that the prediction of Hodge numbers could benefit from multi-task learning.

4 CICYMiner

In the previous section, we showed that a classification task based on Inception modules is effective in learning the Hodge numbers. As the optimization was conducted for $h^{(3,1)}$, rather than an ad hoc structure for each output, the good results motivate further study on learning several Hodge numbers at the same time. In this section, we focus on a regression model for two main reasons. First, in general computations of vector bundle cohomologies, the predictions may not be bounded, thus an inference model has to be able to adapt by learning an approximation function, rather than classification probabilities. Second, previous studies showed that regression models on a similar task were more efficient than classification [21].

Figure 4 shows the schematic of the architecture used in this section. The architecture enables multi-task learning by hard parameter sharing over an initial structure capable of learning a shared representation of the input. This, in general, has proven efficient at increasing the learning power of a single network, rather than differentiating and optimizing several, and to reduce the risk of overfitting [23, 47]. The median layers of the network replicate a similar multi-tasked structure on the same learning objective: in fact, one branch of the sub-structures learning the Hodge numbers is an auxiliary architecture

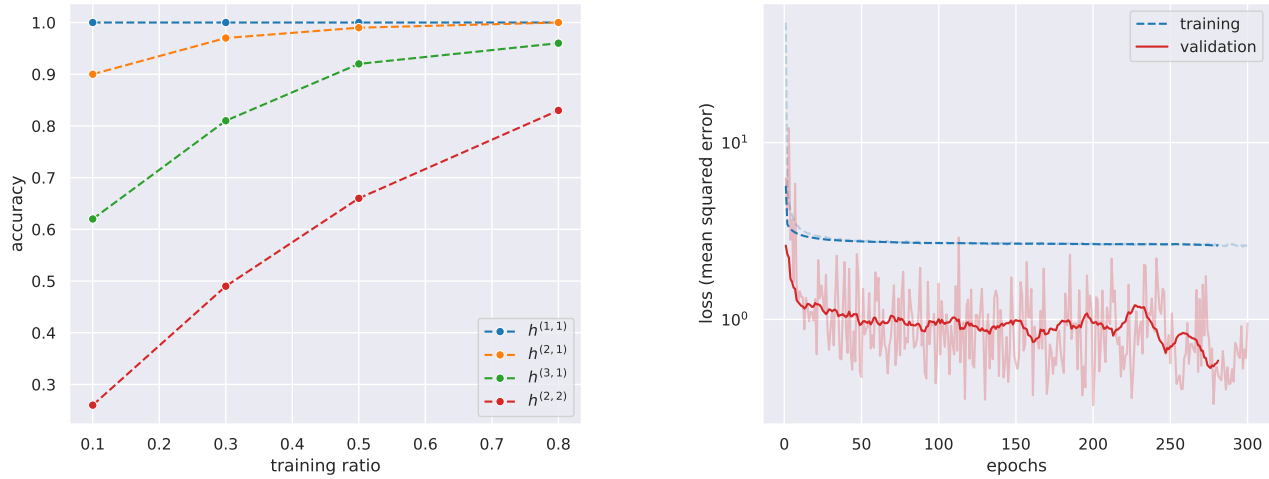


Figure 5: *On the left, we show the final test accuracy of CICYMiner for the four different training ratios. On the right, we present the loss function at* 80 % *training ratio, smoothed with a running average over* 20 *epochs.*

used to reinforce the stability of the representation. No additional regularization was added to the model, apart from a 0.2 dropout rate before the fully connected networks in the auxiliary branches. Such an architecture is thus capable of "mining" richer and more diverse features from a shared representation of the input by using different layer combinations. The model is partly inspired by a recently proposed *DeepMiner* [48] model, used for people re-identification tasks, capable of learning more information by using different branched structures and layers. As such, we refer to our model as *CICYMiner*: we leverage the DeepMiner architecture with the advantages of multi-task learning in order to learn a family of related tasks, which however present complicated and strongly diverse distribution functions (see Figure 1). The role of the auxiliary branches in CICYMiner (see Figure 4) is mainly related to *feature mining*, that is the ability to extract as much information as possible from intermediate representations, in order to guide the learning of the weights during learning. The auxiliary branches have, in fact, slightly different architectures with respect to the main branches, in order to perform different transformation on the inputs. An added value of the auxiliary branches is the duplication of the outputs, which in this multi-task context can improve overall performance, with regard to outlier and overfit control.

4.1 Preprocessing and Evaluation Strategy

We use the same dataset presented for the classification objective in the previous section. Given the strong class imbalance, we select the training set by using a stratified approach on $h^{(2,1)}$ in order to preserve the distribution of the samples. The validation set is then chosen totally at random, using 10 % of the samples. The remaining samples form the test set. We preprocess the input data by simply rescaling the entries of the configuration matrices in the training set to the interval [0, 1]. Matrices in the validation and test sets are rescaled accordingly, using the statistics obtained from the training set.

The outputs of CICYMiner are, in fact, floating point numbers $\tilde{h}^{(i,j)} \in \mathbb{R}^+$, as it is typical in regression tasks. They ultimately need to be rounded to integers to be directly compared with the true values and to compute the accuracy. The distributions of the Hodge numbers have not been rescaled as training led to lower accuracy when this strategy was adopted. The specialised branches of the network are, in fact, deep enough to apply the proper scaling starting from a shared representation and correctly learn the output distribution of the Hodge numbers.

In order to test the robustness and versatility of the network, we choose to keep the outliers in the training set. In multi-task learning architectures, they may strongly affect the behavior of the network and may need robust loss functions during training [49]: this problem is directly addressed in what follows. On the other hand, what repre-

sents an outlier for a certain task, can be valuable information for another [50], hence the choice of keeping the outliers in the training set. Empirically, we also experienced a decrease in accuracy when trying to find a good outlier exclusion strategy.

4.2 Training

In this case, training occurred over a fixed amount of 300 epochs, due to time restrictions on the cluster computing infrastructure. Training takes approximately 5 days on a single NVIDIA V100 GPU. We use the Adam [51] stochastic gradient descent with an initial learning rate of 10^{-3} and a mini-batch size of 64 configuration matrices. Due to the long training time, the optimization was done using a grid search over a reasonable amount of choices of hyperparameters. The network is ultimately made of 10^7 trainable parameters, accounting for both the shared representation and the eight sub-networks learning Hodge numbers and their auxiliary outputs. In terms of typical computer vision multi-task learning, we still deal with a small network: for instance, the original Inception network by Google has 0.7×10^7 parameters for a single classification task [16–18].

We already motivated the choice of keeping the outliers in the training set. We address the arising issues by employing a *Huber* loss function [52]:

$$\mathcal{H}_\delta^{\{k\}}(x) = \begin{cases} \frac{1}{2} \sum_{n=1}^{k} \sum_{i=1}^{N_k} \omega_n \left(x^{(i)}\right)^2, & \left|x^{(i)}\right| \leq \delta \\ \delta \sum_{n=1}^{k} \sum_{i=1}^{N_k} \omega_n \left(\left|x^{(i)}\right| - \frac{\delta}{2}\right), & \left|x^{(i)}\right| > \delta \end{cases} \tag{4.1}$$

where ω_n for $n = 1, 2, \ldots, k$ are the loss weights of the different branches of the CICYMiner, δ is a hyperparameter of the model and $x^{(i)}$ is the residual error of the i-th sample. The choice of the loss turns out to be extremely useful in this regression task, as it behaves as a ℓ_2 loss for small residuals, and it is linear for larger errors. Robustness is thus implemented as a continuous interpolation between the quadratic and linear behaviour of the loss function. This is a solution usually adopted for classification [50] where combinations of ℓ_1, ℓ_2 and Frobenius norm are used for robustness.

In our best implementation, we used $\delta = 1.5$, and loss weights 0.05, 0.3, 0.25 and 0.35 for $h^{(1,1)}$, $h^{(2,1)}$, $h^{(3,1)}$ and $h^{(2,2)}$, respectively (the auxiliary branches use the same values as the principal ones). The learning rate is set to reduce by a factor of 0.3 after 75 epochs without improvements in the total loss of the validation set (as a reference, at 80 % training ratio, this hard reduction mechanism triggered only once between epochs 270 and 300).

4.3 Results

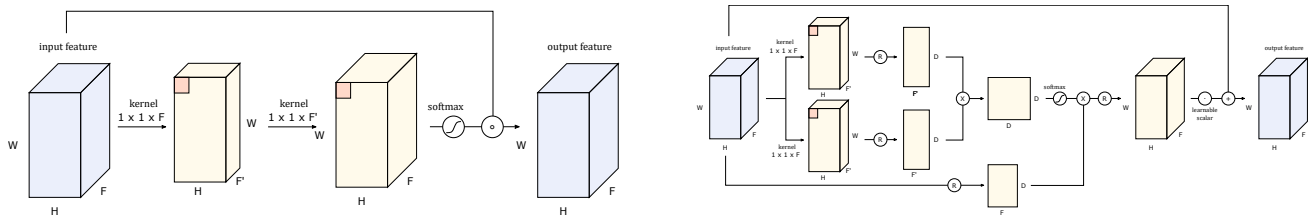
The final results are presented in Figure 5 and in the last row of Table 2. As shown in the learning curve, $h^{(1,1)}$ reaches perfect accuracy with just 10 % of the training data, in alignment with previous attempts [35] and the classification results of the previous section. $h^{(2,2)}$ is in general the most difficult label to train and it is strongly dependent on the training ratio. The network appears to be underfitting the distributions of the Hodge numbers, and validation loss is still decaying after 300 epochs: it would be interesting to run training for longer time, in order to study the behaviour of the network. At a training ratio of 30 % the network reaches perfect accuracy on $h^{(1,1)}$, while $h^{(2,1)}$ gets to 97 %. $h^{(3,1)}$ remains at 81 %, while $h^{(2,2)}$ reaches barely 49 %. Increasing the number of training samples is, in general, beneficial for all Hodge numbers: $h^{(1,1)}$ and $h^{(2,1)}$ reach 100 %, while the accuracy of $h^{(3,1)}$ and $h^{(2,2)}$ rises to 96 % and 83 %, respectively, when the training ratio reaches 80 %. For the first three outputs in Table 2, the regression metrics, Mean Squared Error (MSE) and Mean Absolute Error (MAE), show the ability to effectively learn the discreteness of the Hodge numbers: both metrics show, in fact, values which can be confidently rounded to well defined integer results (i.e. MAE $\ll 0.50$ and MSE $\ll 0.25$).

The good performance of the first three Hodge numbers suggests the possibility to use relations such as the Euler characteristic (3.3), which can be computed from combinatorics, and the linear constraint (3.4). Using the latter to compute $h^{(2,2)}$ leads to an accuracy of 96 % on the test set, using the best results at 80 % training ratio. Using (3.4) and (3.3) together, $h^{(3,1)}$ and $h^{(2,2)}$ can reach perfect accuracy at 80 % training ratio. Using CICYMiner it is therefore possible to compute all four Hodge numbers with 100 % of accuracy.

4.4 Ablation study

CICYMiner introduces new elements, with respect to previous attempts at predicting Hodge numbers of CICYs [21, 35], namely:

1. Huber loss for robustness;



(a) Channel Attention module.

(b) Spatial Attention module.

Figure 6: *Substructures of the attention mechanism used in the ablation study. Here, $\times$ indicates a matrix product along appropriate axes, while $\circ$ is the Hadamard product (element-wise). Reshape operations (R) are also indicated.*

2. auxiliary branches.

In this section, we separately analyse each new aspect, together with other variations of the architecture. Specifically, we analyse the impact of the batch normalization used in the Inception modules. We also address the use of attention mechanisms [53], used in the DeepMiner model, which in our case did not lead to an improvement in accuracy, but rather to a faster training.

We proceed by modifying the backbone structure of CICYMiner. We first introduce the attention mechanism used in [48] for comparison. The Spatial Attention Module (SAM) and CHannel Attention Module (CHAM) are presented in Figure 6: the full attention mechanism is the composition CHAM$\circ$SAM used between each Inception module in the main branch of the task-specific architecture in Figure 4. We also analyse the performance of the model by simply removing the auxiliary branches in the top layers of the network. Then, as opposed to the Huber loss, we test the predictions using the usual MSE used in most regression tasks. We finally change the size and type of the normalization strategy used in the architecture: we first train a network with a mini-batch size of 256 samples, and we then compare the results with a Layer Normalization [54] strategy. Results are summarised in Figure 7 and numerically reported in Table 2. CICYMiner leads to the best overall performance for all four Hodge numbers. The distributions of the residuals, $x^{(i)}$ appearing in the Huber Loss (4.1), show in Figure 8 a homoscedastic behaviour (no correlations between predictions and absolute value of the residuals), which ultimately supports the completeness of the model and its ability to properly predict the four Hodge numbers correctly.

The use of a different loss function, which is not robust against outliers, led to largest drop in accuracy, overall: the difference starts to be consistent even for $h^{(2,1)}$ and $h^{(3,1)}$, which do not present many outliers in Figure 1. The accuracy plummets when considering $h^{(2,2)}$, as expected. The presence of outliers is also evident when increasing the mini-batch size: $h^{(2,2)}$ suffers the largest decrease in accuracy due to such normalisation strategy. At the same time, the introduction of a batch-size independent Layer Normalization strategy, which normalizes each sample over the channel direction rather than the batch dimension, leads to a similar decrease. The presence of outliers seems, therefore, a delicate issue for which the size of the mini-batches plays a relevant role.

A related aspect is represented by the ablation study on the auxiliary branches. As their role is to mine a richer variety of features to stabilise the shared representation, and learn better approximations of the output, the accuracy drops significantly in the case of highly imbalanced distributions. The largest drop impacts $h^{(2,1)}$ which suffers from predictions shifting towards zero. This shows that we indeed need a mechanism to get as much training information as possible through the addition of transformations and auxiliary branches, as in the CICYMiner.

Finally, we analyse the impact of the attention modules: we insert such additional layers to improve the predictions of $h^{(3,1)}$ and $h^{(2,2)}$ only, as other Hodge numbers do not need additional transformations. The results do not strongly differ from the case without the attention modules, though $h^{(2,2)}$ drops by 2 % in accuracy. It therefore seems that the attention modules do not help the predictions in this case, supported by the naive intuition that the configuration matrices do not suggest the development of a sequence model, such as in Natural Language Processing (NLP) or deep learning for video sequences. However, the accuracy reached by the model occurs at around 100 training epochs, rather than 300 as in other cases. The loss function then presents

	$h^{(1,1)}$	$h^{(2,1)}$	$h^{(3,1)}$	$h^{(2,2)}$
+att	**1.00**	0.99	**0.96**	0.81
MSE loss	**1.00**	0.97	0.92	0.50
no aux	**1.00**	0.84	0.92	0.72
bs-256	**1.00**	0.99	0.94	0.65
layer norm	**1.00**	0.99	0.92	0.66
CICYMiner	**1.00**	**1.00**	**0.96**	**0.83**
MSE (10^{-4})	1.3	98	560	6800
MAE (10^{-3})	7.8	19	130	360

Table 2: *Comparison of the accuracy obtained by similar models at* 80 % *training ratio. Regression metrics are also specified for CICYMiner at the same ratio.*

Figure 7: *Summary of the ablation study.*

a slight increase after that. The use of attention modules, together with an early stopping strategy, may therefore significantly cut the training time in this context.

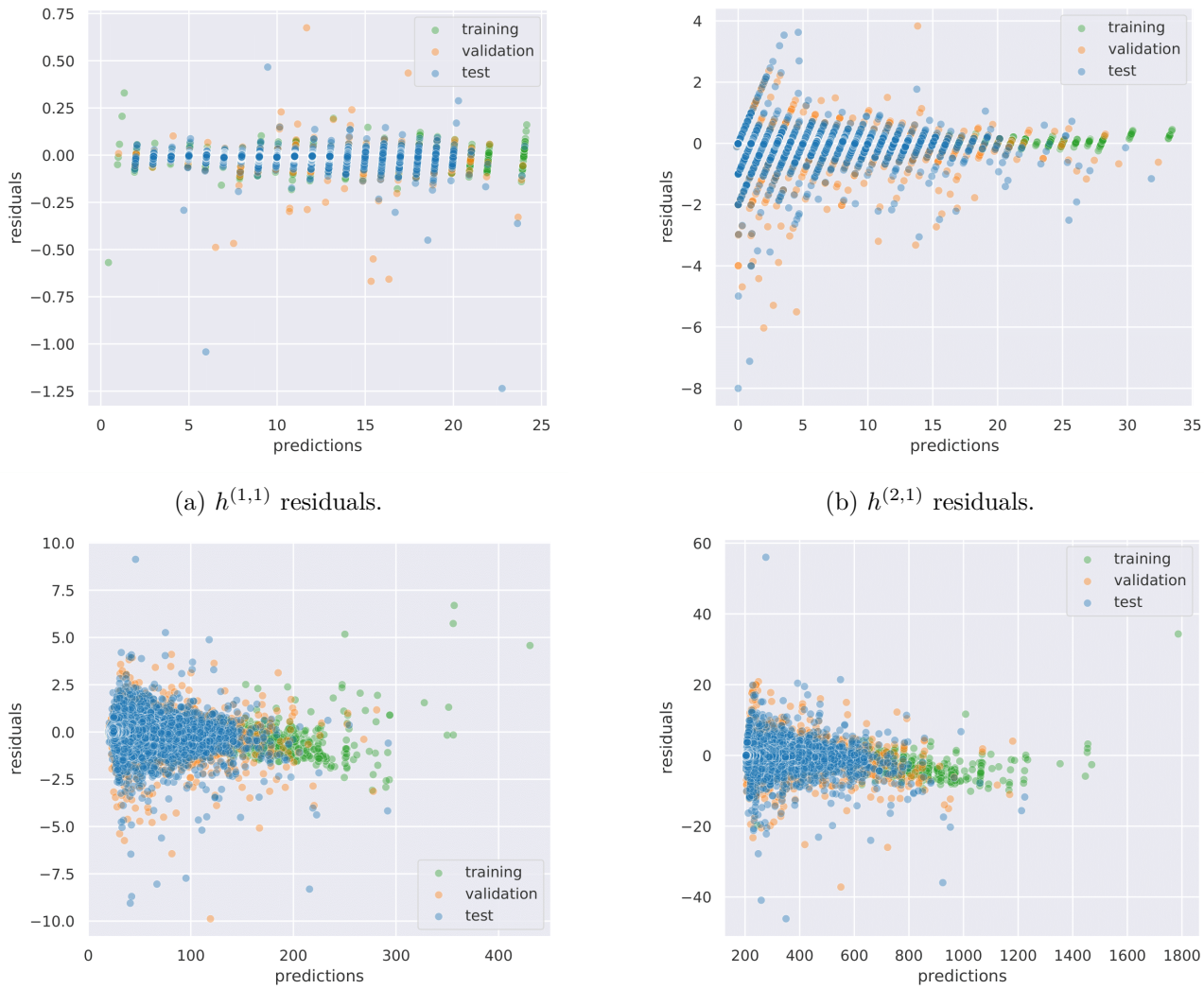
5 Conclusion

In this paper, we were able to show that Inception-based neural networks achieve good accuracy in predicting $h^{(3,1)}$ and $h^{(2,2)}$ and can reach perfect accuracy for the Hodge numbers $h^{(1,1)}, h^{(2,1)}$. Earlier studies using dense architectures were only able to work accurately with $h^{(1,1)}$ [35]. Moreover, we showed that only a fraction of the training data is needed to already obtain promising results. This stands in contrast to earlier studies on CICY three-folds for which it was not possible to accurately predict $h^{(2,1)}$ (the only remaining non-trivial Hodge number in that case). The significant increase in dataset size is responsible for a good part of the increase in performance: the risk of overfitting is strongly reduced and generalization over all configuration matrices is more robust. This is also reflected in the observation that removing the tails of the Hodge number distribution is no longer needed in order to obtain good results. Our main results show that, given the two constraints (3.3) and (3.4) derived from tangent bundle indices, we are able to solve the problem of predicting all Hodge numbers with perfect accuracy.

Our results demonstrate that it is possible to obtain very accurate predictions for the dimension of cohomology groups with only partial training data. We emphasize that the computations of more generic vector bundle cohomologies also satisfy several linear relations and constraints derived from the index, Serre duality or vanishing theorems such as Kodairas. Thus, it is often sufficient to predict a single Hodge number with great accuracy to gain knowledge of all the others. In our experiments, training and validation error align, and we do not observe any significant high variance issues. The high validation and test accuracy suggests that the algorithm produces reliable results, even if it is only trained on partial data, say 30 %. This should open up venues for further investigation into other vector bundle computations.

It is then important to find configurations which yield high accuracy on the validation set. In earlier studies, researchers have used feature enhancement to improve accuracy [22, 35]. Unfortunately, it is not always possible to manipulate the input data via feature engineering in such a way. Adding a relevant monomial basis changes the dimension of the input space in non-trivial ways, such that one has to restrict oneself to a subset of the configuration matrices.

We opted to follow a model-centered approach, common in contemporary machine learning literature, by building a proper architecture with the right amount of parameters. We balance the increased risk of overfitting, due to a larger number of trainable variables, with the natural regularization of multi-task architectures, thus increasing the final accuracy. With its 10^7 parameters, CICYMiner is, given its underlying geometric nature, still a small network with respect to many state-of-art models for computer vision or NLP. Good examples are represented by Inception-Resnet-v2 [55], state-of-the-art in



(a) $h^{(1,1)}$ residuals.

(b) $h^{(2,1)}$ residuals.

(c) $h^{(3,1)}$ residuals.

(d) $h^{(2,2)}$ residuals.

Figure 8: *Residual plots at* 80 % *training ratio.*

single-task image classification with 5.6×10^7 parameters and a long training time on 20 NVidia Kepler GPUs, and GPT-3 [56], state-of-the-art NLP model, with more than 175×10^9 model parameters. In fact, recent research suggests that neural networks often admit power-scaling laws with dataset size and model parameters [57]. It can also be noted that increasing the capacity of the model may be beneficial to the overall performance [58]. However, the geometric and physical interpretability might then become quite complicated and involved, hence the suggestion to constrain the complexity of the CICYMiner architecture. It would be interesting to observe how far one can improve the accuracy of $h^{(3,1)}$ and $h^{(2,2)}$ by using larger networks or adding more data samples to the dataset, or even by just prolonging the training time on multiple GPUs. Additional data samples can in principle be easily generated via (in-)effective splits of the already existing configuration matrices. These redundant matrices had been discarded when compiling the initial dataset [12].

As a conclusion, our paper builds further the case for using deep learning in algebraic geometry by demonstrating that an appropriate neural network architecture can predict accurately Hodge numbers of CICY. Moreover, since algebraic geometry uses datasets which are not of the type usually encountered in usual machine learning applications, our results extend their range of applications.

Acknowledgements

RS is funded in part by the Swedish Research Council (VR) under grant numbers 2016-03873, 2016-03503, and 2020-03230. RS is grateful for

financial support from the Liljewalch scholarship. HE is funded by the European Union's Horizon 2020 research and innovation program under the Marie Skłodowska-Curie grant agreement No 891169. HE is also supported by the National Science Foundation under Cooperative Agreement PHY-2019786 (The NSF AI Institute for Artificial Intelligence and Fundamental Interactions, http://iaifi.org/). The work of RF is supported by a joint programme (PTC) between the *Direction des énergies* and the *Direction de la recherche tecnologique* of the CEA Paris–Saclay. Computations were in part enabled by resources provided by the Swedish National Infrastructure for Computing (SNIC) at the HPC cluster *Tetralith*, partially funded by the Swedish Research Council through grant agreement no. 2018-05973, and the *FactoryIA* supercomputer, financially supported by the Ile-de-France Regional Council.

References

[1] F. Ruehle. "Data science applications to string theory." *Phys. Rept.* 839 (2020), pp. 1–117. DOI: 10.1016/j.physrep.2019.09.005.

[2] F. Denef and M. R. Douglas. "Computational Complexity of the Landscape I." *Annals of Physics* 322.5 (2007), pp. 1096–1142. DOI: 10.1016/j.aop.2006.07.013. arXiv: hep-th/0602072.

[3] J. Halverson and F. Ruehle. "Computational Complexity of Vacua and Near-Vacua in Field and String Theory." *Phys. Rev. D* 99.4 (2019), p. 046015. DOI: 10.1103/PhysRevD.99.046015. arXiv: 1809.08279 [hep-th].

[4] W. Taylor and Y.-N. Wang. "The F-theory geometry with most flux vacua." *JHEP* 12 (2015), p. 164. DOI: 10.1007/JHEP12(2015)164. arXiv: 1511.03209 [hep-th].

[5] J. Halverson, B. Nelson, and F. Ruehle. "Branes with Brains: Exploring String Vacua with Deep Reinforcement Learning." *JHEP* 06 (2019), p. 003. DOI: 10.1007/JHEP06(2019)003. arXiv: 1903.11616 [hep-th].

[6] M. Larfors and R. Schneider. "Explore and Exploit with Heterotic Line Bundle Models." *Fortsch. Phys.* 68.5 (2020), p. 2000034. DOI: 10.1002/prop.202000034. arXiv: 2003.04817 [hep-th].

[7] Y.-H. He. "Calabi-Yau Spaces in the String Landscape." *Oxford University Press* Oxford Research Encyclopedia of Physics (2020). DOI: 10.1093/acrefore/9780190871994.013.60. arXiv: 2006.16623.

[8] P. Candelas, A. M. Dale, C. A. Lutken, and R. Schimmrigk. "Complete Intersection Calabi-Yau Manifolds." *Nucl. Phys.* B298 (1988), p. 493. DOI: 10.1016/0550-3213(88)90352-5.

[9] P. S. Green, T. Hubsch, and C. A. Lutken. "All Hodge Numbers of All Complete Intersection Calabi-Yau Manifolds." *Class. Quant. Grav.* 6 (1989), pp. 105–124. DOI: 10.1088/0264-9381/6/2/006.

[10] L. B. Anderson, X. Gao, J. Gray, and S.-J. Lee. "Fibrations in CICY Threefolds." *Journal of High Energy Physics* 2017.10 (2017). DOI: 10.1007/JHEP10(2017)077. arXiv: 1708.07907.

[11] M. Kreuzer and H. Skarke. "Complete classification of reflexive polyhedra in four-dimensions." *Adv. Theor. Math. Phys.* 4 (2002), pp. 1209–1230. DOI: 10.4310/ATMP.2000.v4.n6.a2. arXiv: hep-th/0002240.

[12] J. Gray, A. S. Haupt, and A. Lukas. "All Complete Intersection Calabi-Yau Four-Folds." *JHEP* 07 (2013), p. 070. DOI: 10.1007/JHEP07(2013)070. arXiv: 1303.1832 [hep-th].

[13] J. Gray, A. S. Haupt, and A. Lukas. "Topological Invariants and Fibration Structure of Complete Intersection Calabi-Yau Four-Folds." *JHEP* 09 (2014), p. 093. DOI: 10.1007/JHEP09(2014)093. arXiv: 1405.2073 [hep-th].

[14] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei. "ImageNet Large Scale Visual Recognition Challenge." *International Journal of Computer Vision (IJCV)* 115.3 (2015), pp. 211–252. DOI: 10.1007/s11263-015-0816-y.

[15] A. Krizhevsky, I. Sutskever, and G. E. Hinton. "ImageNet Classification with Deep Convolutional Neural Networks." *Commun. ACM* 60.6 (2017), 84–90. DOI: 10.1145/3065386. URL: https://doi.org/10.1145/3065386.

[16] C. Szegedy, Wei Liu, Yangqing Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. "Going Deeper with Convolutions." In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015, pp. 1–9. DOI: 10.1109/CVPR.2015.7298594.

[17] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. "Rethinking the Inception Architecture for Computer Vision." In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, pp. 2818–2826. DOI: 10.1109/CVPR.2016.308. arXiv: 1512.00567.

[18] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi. "Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning." In: *31st AAAI Conference on Artificial Intelligence*. AAAI'17. AAAI Press, 2017, pp. 4278–4284. arXiv: 1602.07261.

[19] K. He, X. Zhang, S. Ren, and J. Sun. "Deep residual learning for image recognition." In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778. DOI: 10.1109/CVPR.2016.90. arXiv: 1512.03385 [cs.CV].

[20] Y.-H. He. "Deep-Learning the Landscape" (2017). arXiv: 1706.02714 [hep-th].

[21] H. Erbin and R. Finotello. "Inception neural network for complete intersection Calabi–Yau 3-folds." *Mach. Learn. Sci. Tech.* 2.2 (2021), 02LT03. DOI: 10.1088/2632-2153/abda61. arXiv: 2007.13379 [hep-th].

[22] H. Erbin and R. Finotello. "Machine learning for complete intersection Calabi-Yau manifolds: a methodological study" (2020). arXiv: 2007.15706 [hep-th].

[23] R. Caruana. "Multitask Learning: A Knowledge-Based Source of Inductive Bias." In: *Proceedings of the Tenth International Conference on Machine Learning*. Morgan Kaufmann, 1993, pp. 41–48. DOI: 10.1016/b978-1-55860-307-3.50012-5.

[24] T. Standley, A. R. Zamir, D. Chen, L. Guibas, J. Malik, and S. Savarese. *Which Tasks Should Be Learned Together in Multi-task Learning?* 2020. arXiv: 1905.07553 [cs.CV].

[25] T. pandas development team. *pandas-dev/pandas: Pandas*. Version latest. 2020. DOI: 10.5281/zenodo.3509134. URL: https://doi.org/10.5281/zenodo.3509134.

[26] Wes McKinney. "Data Structures for Statistical Computing in Python." In: *Proceedings of the 9th Python in Science Conference*. Ed. by Stéfan van der Walt and Jarrod Millman. 2010, pp. 56–61. DOI: 10.25080/Majora-92bf1922-00a.

[27] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant. "Array programming with NumPy." *Nature* 585.7825 (2020), pp. 357–362. DOI: 10.1038/s41586-020-2649-2. URL: https://doi.org/10.1038/s41586-020-2649-2.

[28] J. D. Hunter. "Matplotlib: A 2D graphics environment." *Computing in Science & Engineering* 9.3 (2007), pp. 90–95. DOI: 10.1109/MCSE.2007.55.

[29] M. L. Waskom. "seaborn: statistical data visualization." *Journal of Open Source Software* 6.60 (2021), p. 3021. DOI: 10.21105/joss.03021. URL: https://doi.org/10.21105/joss.03021.

[30] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: https://www.tensorflow.org/.

[31] K. Bull, Y.-H. He, V. Jejjala, and C. Mishra. "Machine Learning CICY Threefolds." *Phys. Lett. B* 785 (2018), pp. 65–72. DOI: 10.1016/j.physletb.2018.08.008. arXiv: 1806.03121 [hep-th].

[32] K. Bull, Y.-H. He, V. Jejjala, and C. Mishra. "Getting CICY High." *Phys. Lett. B* 795 (2019), pp. 700–706. DOI: 10.1016/j.physletb.2019.06.067. arXiv: 1903.03113 [hep-th].

[33] S. Krippendorf and M. Syvaeri. "Detecting symmetries with neural networks." *Machine Learning: Science and Technology* 2.1 (Dec. 2020), p. 015010. DOI: 10.1088/2632-2153/abbd2d. arXiv: 2003.13679 [physics.comp-ph+].

[34] Y.-H. He and S.-J. Lee. "Distinguishing Elliptic Fibrations with AI." *Physics Letters B* 798 (2019), p. 134889. DOI: 10.1016/j.physletb.2019.134889. arXiv: 1904.08530.

[35] Y.-H. He and A. Lukas. "Machine Learning Calabi-Yau Four-folds." *Phys. Lett. B* 815 (2021), p. 136139. DOI: 10.1016/j.physletb.2021.136139. arXiv: 2009.02544 [hep-th].

[36] M. Demirtas, L. McAllister, and A. Rios-Tascon. "Bounding the Kreuzer-Skarke Landscape" (2020). arXiv: 2008.01730 [hep-th].

[37] M. Bies, M. Cvetič, R. Donagi, L. Lin, M. Liu, and F. Ruehle. "Machine Learning and Algebraic Approaches towards Complete Matter Spectra in 4d F-theory" (2020). arXiv: 2007.00009 [hep-th].

[38] F. Ruehle. "Evolving neural networks with genetic algorithms to study the String Landscape." *JHEP* 08 (2017), p. 038. DOI: 10.1007/JHEP08(2017)038. arXiv: 1706.07024 [hep-th].

[39] M. Larfors and R. Schneider. "Line bundle cohomologies on CICYs with Picard number two." *Fortsch. Phys.* 67.12 (2019), p. 1900083. DOI: 10.1002/prop.201900083. arXiv: 1906.00392 [hep-th].

[40] D. Klaewer and L. Schlechter. "Machine Learning Line Bundle Cohomologies of Hypersurfaces in Toric Varieties." *Phys. Lett. B* 789 (2019), pp. 438–443. DOI: 10.1016/j.physletb.2019.01.002. arXiv: 1809.02547 [hep-th].

[41] C. R. Brodie, A. Constantin, R. Deen, and A. Lukas. "Machine Learning Line Bundle Cohomology." *Fortsch. Phys.* 68.1 (2020), p. 1900087. DOI: 10.1002/prop.201900087. arXiv: 1906.08730 [hep-th].

[42] C. R. Brodie, A. Constantin, R. Deen, and A. Lukas. "Index Formulae for Line Bundle Cohomology on Complex Surfaces." *Fortschritte der Physik* 68.2 (2020), p. 1900086. DOI: 10.1002/prop.201900086. arXiv: 1906.08769.

[43] A. Constantin and A. Lukas. "Formulae for Line Bundle Cohomology on Calabi-Yau Threefolds." *Fortsch. Phys.* 67.12 (2019), p. 1900084. DOI: 10.1002/prop.201900084. arXiv: 1808.09992 [hep-th].

[44] S. Falkner, A. Klein, and F. Hutter. "BOHB: Robust and Efficient Hyperparameter Optimization at Scale." In: *Proceedings of the 35th International Conference on Machine Learning*. Ed. by J. Dy and A. Krause. Vol. 80. Proceedings of Machine Learning Research. Stockholmsmässan, Stockholm Sweden: PMLR, 2018, pp. 1437–1446. URL: http://proceedings.mlr.press/v80/falkner18a.html.

[45] L. Li, K. Jamieson, G. DeSalvo, A. Rostamizadeh, and A. Talwalkar. "Hyperband: A Novel Bandit-Based Approach to Hyperparameter Optimization." *Journal of Machine Learning Research* 18.185 (2018), pp. 1–52. URL: http://jmlr.org/papers/v18/16-558.html.

[46] S. Ioffe and C. Szegedy. "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift." *CoRR* abs/1502.03167 (2015). arXiv: 1502.03167. URL: http://arxiv.org/abs/1502.03167.

[47] J. Baxter. "A Bayesian/Information Theoretic Model of Learning to Learn via Multiple Task Sampling." *Machine Learning* 28.1 (July 1, 1997), pp. 7–39. DOI: 10.1023/A:1007327622663. URL: https://doi.org/10.1023/A:1007327622663.

[48] A. Benzine, M. El Amine Seddik, and J. Desmarais. "Deep Miner: A Deep and Multi-branch Network which Mines Rich and Diverse Features for Person Re-identification," arXiv:2102.09321 (2021), arXiv:2102.09321. arXiv: 2102.09321 [cs.CV].

[49] R. Zhang, H. Zhang, and X. Li. "Robust Multi-Task Learning With Flexible Manifold Constraint." *IEEE Transactions on Pattern Analysis and Machine Intelligence* 43.6 (2021), pp. 2150–2157. DOI: 10.1109/TPAMI.2020.3007637.

[50] Y. Zhang and Q. Yang. "A Survey on Multi-Task Learning." *IEEE Transactions on Knowledge and Data Engineering* (2021), pp. 1–1. DOI: 10.1109/TKDE.2021.3070203.

[51] D. P. Kingma and J. Ba. *Adam: A method for stochastic optimization*. 2014. arXiv: 1412.6980 [cs.LG].

[52] P. J. Huber. "Robust Estimation of a Location Parameter." *The Annals of Mathematical Statistics* 35.1 (1964), pp. 73 –101. DOI: 10.1214/aoms/1177703732. URL: https://doi.org/10.1214/aoms/1177703732.

[53] D. Bahdanau, K. Cho, and Y. Bengio. "Neural Machine Translation by Jointly Learning to Align and Translate," arXiv:1409.0473 (2016). arXiv: 1409.0473 [cs.CL].

[54] J. L. Ba, J. R. Kiros, and G. E. Hinton. *Layer Normalization*. 2016. arXiv: 1607.06450 [stat.ML].

[55] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. Alemi. *Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning*. 2016. arXiv: 1602.07261 [cs.CV].

[56] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei. "Language Models are Few-Shot Learners." *CoRR*

abs/2005.14165 (2020). arXiv: 2005.14165.
URL: https://arxiv.org/abs/2005.14165.

[57] Y. Bahri, E. Dyer, J. Kaplan, J. Lee, and U. Sharma. "Explaining Neural Scaling Laws." *CoRR* abs/2102.06701 (2021). arXiv: 2102.06701.
URL: https://arxiv.org/abs/2102.06701.

[58] M. Belkin, D. Hsu, S. Ma, and S. Mandal. "Reconciling modern machine-learning practice and the classical bias–variance trade-off." *Proceedings of the National Academy of Sciences* 116.32 (2019), pp. 15849–15854.