Inception neural network for complete intersection Calabi–Yau 3-folds

IOP Publishing Mach. Learn.: Sci. Technol. 2 (2021) 02LT03 https://doi.org/10.1088/2632-2153/abda61

MACHINE LEARNING Science and Technology

RECEIVED
20 November 2020

REVISED
22 December 2020

ACCEPTED FOR PUBLICATION
11 January 2021

PUBLISHED
2 March 2021

LETTER

Inception neural network for complete intersection Calabi–Yau 3-folds

H Erbin and R Finotello

Dipartimento di Fisica, Università di Torino and I.N.F.N.—sezione di Torino, via P Giuria 1, Torino I-10125, Italy

E-mail: erbin@to.infn.it and riccardo.finotello@to.infn.it

Keywords: Calabi–Yau manifold, deep learning, string theory compactification, algebraic topology

Abstract

We introduce a neural network inspired by Google's Inception model to compute the Hodge number $h^{1,1}$ of complete intersection Calabi–Yau (CICY) 3-folds. This architecture improves largely the accuracy of the predictions over existing results, giving already 97% of accuracy with just 30% of the data for training. Accuracy climbs to 99% when using 80% of the data for training. This proves that neural networks are a valuable resource to study geometric aspects in both pure mathematics and string theory.

1. Introduction

The last few years have witnessed the uprising of *deep learning* as a very efficient method to elaborate, process and learn patterns in data [1]. While the underlying ideas behind neural networks are not recent [2, 3], larger databases, and computational capabilities together with new techniques have led to deep learning pervading most fields of scientific research and industrial development.

Understanding geometrical structures is an emerging application of machine learning, referred to as *geometric deep learning* [4, 5] when neural networks are used. This is an important problem for different fields: for example in the industry (e.g. for 3*d* modelling of objects), computer science (e.g. for gradient optimisation [6]), pure mathematics, and theoretical physics. For this reason it is crucial to adapt existing techniques or to design new ones if needed.

In this paper we focus on the computation of the Hodge number $h^{1,1}$ for complete intersection Calabi–Yau (CICY) 3-folds [7]. This is a challenging mathematical problem *per se* because traditional methods from algebraic topology lead to complicated algorithms, without closed-form expressions in most cases. Machine learning techniques give the possibility to speed up computations and to obtain hints to better understand the mathematical structures. Moreover, Calabi–Yau manifolds, beyond being important mathematical objects, also have a distinguished role in string theory as they are needed to describe the compactified dimensions [8]. In particular the general properties of the four-dimensional effective field theory are completely determined by the topology. Given the complexity of the space of string vacua, developing faster and efficient computational techniques is essential in the search of the Standard Model (or an extension compatible with experiments) within string theory at low energy. Finally, this type of objects is quite remote from typical data considered in machine learning, which calls for an evaluation of existing techniques in this context and, if they are not sufficient, the development of new approaches.

The CICY 3-folds are appropriate for this task: since they have been completely classified [9–11], they provide a simple playground where it is possible to test different machine learning techniques. The goal of this paper is to continue the study started in [12, 13], which used machine learning techniques to compute $h^{1,1}$ (see also [14–16] for other papers on CICY 3-folds). Related applications on the study of cohomology groups are [17–19]. For an introduction to machine learning and its applications to string theory, we refer to the excellent review [20].

Most breakthroughs in AI and industrial applications of deep learning usually followed the discovery of a new network model. This is particularly true in computer vision where convolutional, Inception and residual networks [3, 21–24] have been major cornerstones. In this work we introduce an alternative version of

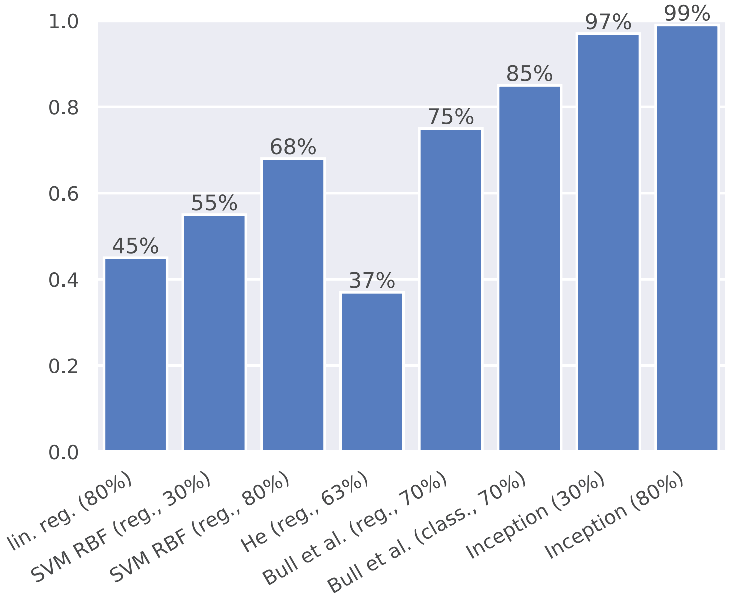

Figure 1. Accuracy reached by different models. The percentage in parenthesis indicates the ratio of training data. 'He' refers to [12], 'Bull *et al*' to [13]. Each model except the Inception one keeps outliers in the training set (the effects is marginal in linear regression and support-vector machines).

Google's Inception network [21–23] (see [20] for a review) to predict $h^{1,1}$ from the configuration matrix of CICY 3-folds. Using 30% of training data, we reach close to 97% accuracy on the predictions, improving by a large measure previous results [12, 13] with much less training data and parameters (≈234 000). Using 80% for the data for training we obtain 99% accuracy.

This must be compared with the following accuracies: 37% (regression, fully connected network, ≈280 000 parameters, 63% training data) in [12], 75% (regression, fully connected network, ≈1 580 000 parameters, 70% training data) and 85% (classification, convolutional network, 70% training data) in [13] (figure 1). More generally, we found that the Inception-like network performs much better than any other machine learning algorithm, even after feature engineering [25]: the best algorithm after neural networks is a support-vector machine (SVM) with a radial basis function (RBF) kernel, which reaches 68% accuracy with 80% of training data [13, 25]. This shows that neural networks are able to make accurate predictions for Hodge numbers, as long as the correct architecture is found. This opens the door to new applications to theoretical physics and mathematics which may lead to even further progress.

The code is written in Python and relies on the following packages: `scikit-learn` [26], `tensorflow` [27] (and its high level API, `keras` [28]) and the `scipy` ecosystem for visualisation and computations [29].

2. General setup

The dataset [9, 10] is made of 7890 CICY 3-folds, described by their configuration matrices and their topological properties, including the Hodge numbers $h^{1,1}$ and $h^{2,1}$. We focus on predicting the Hodge number $h^{1,1} \in \mathbb{N}$, which lies in the closed interval $[0,\ 19]$ with 18 distinct values (with $h^{1,1} = 17, 18$ not present), from the configuration matrix:

$$\left[\begin{array}{c|ccc} \mathbb{P}^{n_1} & a_1^1 & \cdots & a_k^1 \\ \vdots & \vdots & \ddots & \vdots \\ \mathbb{P}^{n_m} & a_1^m & \cdots & a_k^m \end{array}\right], \quad a_\alpha^r \in \mathbb{N} \quad \longrightarrow \quad h^{1,1} \in \mathbb{N}. \tag{1}$$

The configuration matrix describes the CICY as the intersection of k hypersurfaces, characterised by a system of homogeneous polynomial equations, inside the ambient space $\mathbb{P}^{n_1} \times \cdots \times \mathbb{P}^{n_m}$, where m denotes the number of complex projective spaces. The coefficients a_α^r of the matrix denote the power of the coordinates of each projective space entering each polynomial equation. This data is sufficient to characterise the topology. For more information on CICY we refer the reader to the literature [9, 10, 30–33].

We consider the problem as a regression task and not as a classification task even if the outputs are integers. Indeed the latter requires knowledge of all possible Hodge numbers which can appear and prevents

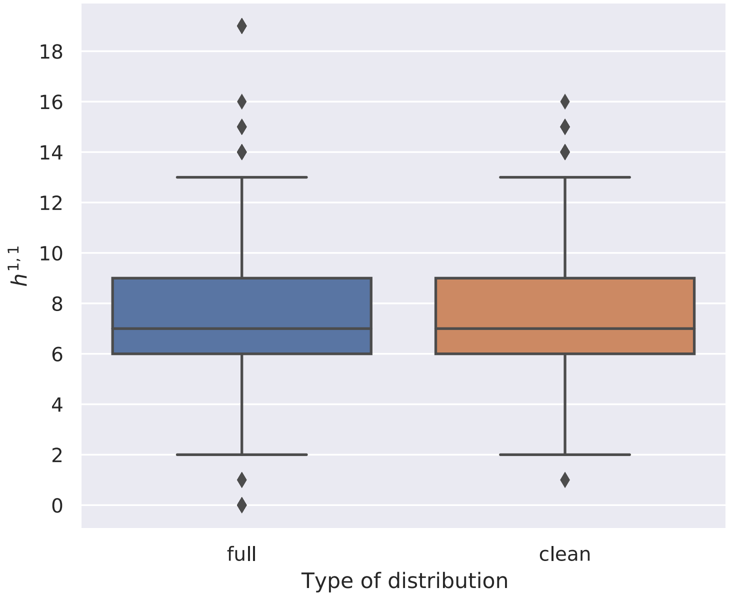

Figure 2. Whisker plot for the distribution of $h^{1,1}$ before (*full*) and after (*clean*) removing the outliers. The coloured boxes highlight the interval between the first and third quartiles while the internal horizontal line represents the median value. The 'whiskers' delimit the interquartile range, while isolated points mark the remaining outliers.

any extrapolation, which is not desirable in the current context. Since regression algorithms output a real number, it is necessary to map predictions to integers before comparing with the real values.

The dataset is split into three subsets: one for training (used to learn the optimal model weights with gradient descent), one for validation (hyperparameter tuning and early stopping in neural networks), and one for testing.

In the following section we discuss a few properties of the dataset which play an important role in the training of the neural network introduced in the next section.

2.1. Exploratory data analysis

The first step before writing the neural network is to better understand the data. Displaying the distribution of the Hodge numbers in the whisker plot in the left side of figure 2, one finds the presence of outliers at small and high Hodge numbers. Outliers can strongly impede the learning process of most algorithms and they must be handled with care. In this paper we obtained the best accuracy by simply removing them from the training data (but keeping them in the test set).

The outliers fall into two classes. First, the product spaces are recognisable by having vanishing Hodge numbers $h^{1,1} = h^{2,1} = 0$ and a block-diagonal configuration matrix.[3] Second we deal with manifolds with high Hodge numbers. We keep only manifolds such that $h^{1,1} \in [1, 16]$ and $h^{2,1} \in [15, 86]$ in the training data. Over the full dataset only 39 samples are excluded, or 0.49%. Hence training samples are taken as a subset of the distribution given in the right side of figure 2. We expect systematical errors on test samples among outliers but they are too few to drastically impact the accuracy.

2.2. Baseline

It is important to design a simple baseline model to quantify the gain of using a neural network. Here we consider a linear regression with ℓ_1 regularisation with parameter 2×10^{-4} and without intercept. Integers are obtained by flooring the predictions to the next lower integers. We obtain 47%–51% accuracy using 20%–80% of the data for training.

Moreover a simple analysis [25] shows that the number of projective spaces m (number of rows of the matrix) is an important feature. Performing a linear regression with ℓ_1 weight of 1.0, we obtain 63% of accuracy. This is related to a known mathematical result [32] stating that the so-called favourable matrices have $h^{1,1} = m$ (in the dataset from [9, 10], there are 4874 favourable matrices). If it had not been known, the linear regression could have led to conjecture that this formula—indeed, conjecture generation is another distinguished use of machine learning techniques for theoretical physics [34, 19]. Note that SVM with RBF

[3] Note that $h^{1,1} = 0$ is not the actual value of $h^{1,1}$ but indicates merely that the CICY is factorisable into products of tori and K3 surfaces.

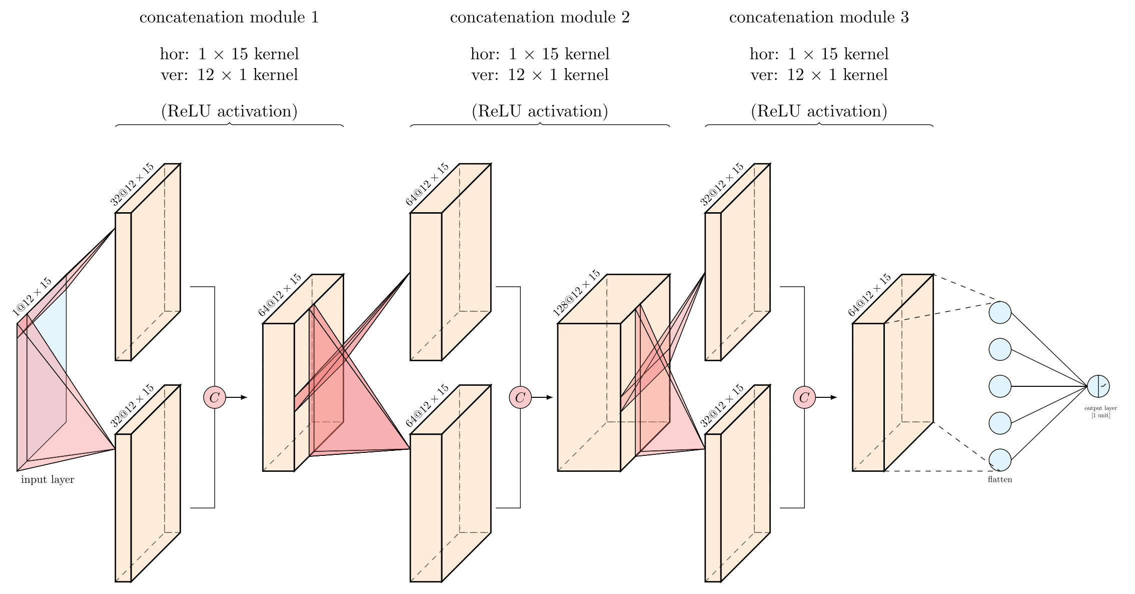

Figure 3. Schematic figure of the Inception model used to predict $h^{1,1}$ from configuration matrices.

kernel is the best ML algorithm outside neural networks but improves only marginally over linear regression (figure 1) [25].

3. Inception neural network

In this section we introduce a new deep learning architecture capable of predicting accurately $h^{1,1}$ from the configuration matrix of the CICY manifolds. Though different both in purpose and in definition, the model is inspired by Google's *Inception network* [21–23]. This deep neural network uses Inception modules performing different concurrent convolutional operations to enhance, process, and rearrange its input (in Google's case, images to be classified over 1000 classes in the *ImageNet* repository). This architecture encountered great success as it obtained results much better than any other machine learning algorithm until then. Modifications of the original model brought even higher accuracy and enhancement of computer vision capabilities. We refer the reader to [20] for a review of Inception networks.

We arrived at this network by going through neural network architectures used in computer vision. Indeed, the configuration matrix being a matrix of integers, it resembles an image with one channel. Since a sequential convolutional network does not reach a sufficient accuracy and needs a lot of training data, the Inception is the next natural step. Its structure has been guided by the form of the configuration matrix (see subsection 3.4 for more details).

Adapting this network to our problem, we obtain close to 100% accuracy already by training with only 30% of the data, which is much higher than existing results [12, 13]. A more general machine learning analysis of this problem will appear in [25].

3.1. Architecture

The architecture is schematically depicted in figure 3: it is divided into three *Inception* modules followed by an output layer with a single unit for the prediction of the Hodge number.

The first layer takes the configuration matrices as input, which are represented as tensors of shape (12, 15, 1) (matrices with a single channel). Next two parallel convolutions (shown in red in figure 3) are performed: one over the rows (12 × 1 kernel, processing each projective space at a time) and one over the columns (1 × 15 kernel, processing each equation of the polynomial system at a time). The outputs of both layers are concatenated together over the channel dimension. These two steps form an Inception module, which is repeated three times in total, with respectively 32, 64, and 32 filters. All convolutional layers and the final layer are followed by a ReLU activation function and each concatenation by a batch normalisation with momentum 0.99. A dropout layer with a rate 0.2 after the last Inception module and before flattening the results, to connect it to the final output layer. Finally all layers have ℓ_1 and ℓ_2 regularisation, respectively, with weights 10^{-4} and 10^{-3}. The network has ≈234 000 parameters, which is less than previous proposals [12, 13]. This is achieved by using only convolutional layers with relatively small kernels.

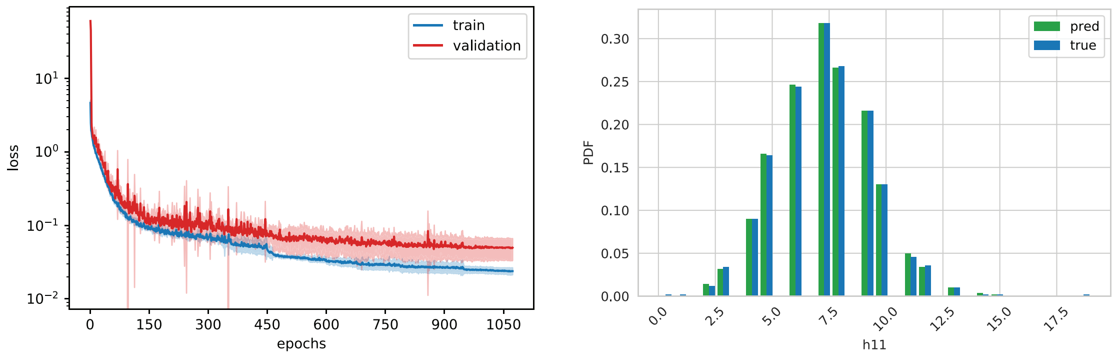

Figure 4. On the left we show the evolution of the loss evaluated on the training (blue curve) and evaluation (red curve) sets during training with 80% training data (the coloured area denotes the 1σ region). On the right we present the distribution of the real and predicted values for the Inception network (80% training data).

Note that there are no pooling layers. Convolutions use `same` as padding value which allows us to keep the same size (12, 15) as the input. The output layer is followed by a ReLU activation function which forces the result to be positive, as it should be for Hodge numbers.

This architecture has two evident advantages over a fully connected (FC) network or even a more classical convolutional structure. First the network concurrently learns different representations and automatically combines them in more complex representations. Second the number of parameters is extremely restricted.

3.2. Training and validation strategy

We use a *holdout* validation strategy: the dataset is divided into three subsets for training (gradient descent to optimise the neural network's weights), validation (early stopping, and hyperparameter tuning) and testing purposes (final assessment of our model). We retain respectively 80% of all samples for training, 10% for validation, and 10% for testing.

Before feeding the configuration matrix to the neural network, we first remove the outliers as discussed previously. We have tried to rescale the matrix by dividing by the highest entry (5), but this does not bring any significant improvement.

We did not use any data augmentation. Adding matrices with permutations of rows and columns seem to decrease the performance of the neural network: one possible explanation is that matrix components are ordered lexicographically [9]. Moreover, we did not generated more matrices using mathematical equivalences [9] since the final accuracy is high enough.

Hyperparameter tuning (number of Inception modules and filters, dropout rate, etc) has been performed by hand by evaluating several models on the validation set. After finding the appropriate architecture, described in the previous subsection, we have also evaluated the accuracy by training with 30% and 50% of the data (keeping always 10% for the validation set, necessary for early stopping).

The neural network is trained using the *Adam* [35] optimiser with default parameters, initial learning rate 10^{-3} and a batch size of 32. We use the mean squared error of the predictions as a loss function. The learning rate is reduced by a factor of 0.3 when the validation loss does not decrease during 75 epochs. We also use early stopping: the network is trained until the validation loss does not decrease for 200 epochs, restoring the weights associated with the lowest validation loss.

Predictions are obtained by averaging the results of five neural networks (bagging), which allows us to reduce the variance and obtain the standard deviation of the results. Since predictions are real numbers at this point, they are rounded to the closest integers before comparing them with the real value. The performance of the model is measured by the accuracy, which is the ratio of predictions matching exactly the real values.

Finally we will also provide learning curves for the neural network described in the previous section. For this we split the dataset into training and validation subsets with different relative ratios and we compute the accuracy on both sets after training. In each case we keep 10% of the training data for early stopping. Exception made for this, the rest of the setup is the same.

3.3. Results

In figure 4, we show the evolution of the training and validation loss (mean squared error) during training. Curiously the mean absolute error is smaller for the validation set.

The agreement between the predictions and real values is excellent on the test fold. The distributions are displayed in figure 4. The results at different ratios of training data are given in table 1, where we also display

Table 1. Accuracy for the Inception neural network for different sizes of the training dataset, with standard deviations between 0.1% and 0.5%. Results obtained for other models are added for comparison: fully connected network [13] (read from figure 4), convolutional network [25]. See also figure 1.

Training data	Fully connected	Convolution	Inception
80%	≈77%	92.5%	98.7%
50%	≈74%	84.9%	98.3%
30%	≈68%	78.5%	97.6%

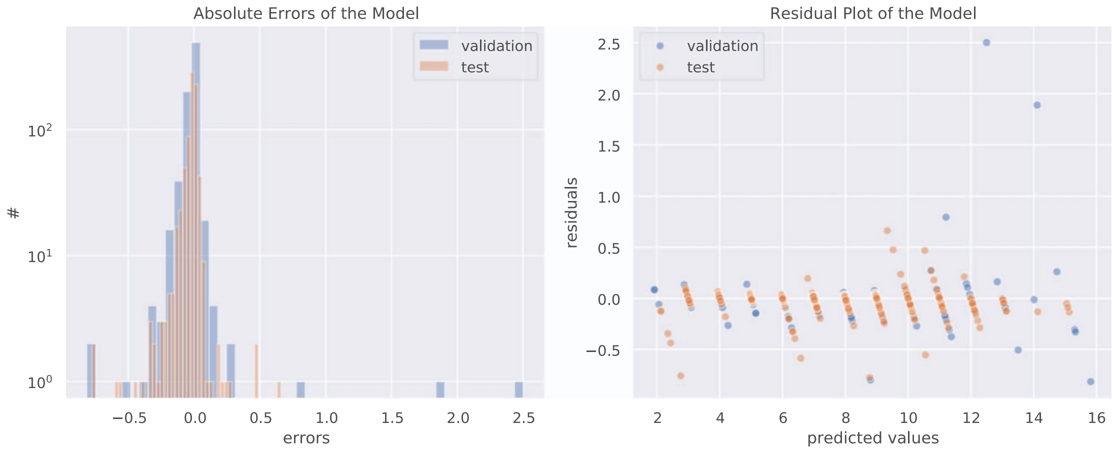

Figure 5. Difference between the true values of $h^{1,1}$ and their predictions seen as a univariate distribution (on the left) and as a function of the predicted value (on the right): the histogram shows the distribution and extension in values of the difference between true values and predictions, while the scatter plot exhibits the constant variance of the residual error (the network performs equally well over the entire range of $h^{1,1}$).

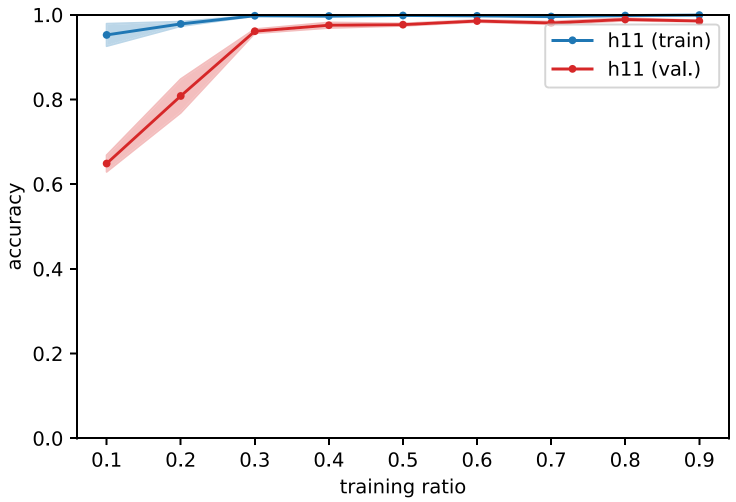

Figure 6. Learning curve of the Inception network. The coloured area denotes the 1σ region.

the accuracy for other regression models: the fully connected network from [13] and an improved sequential convolutional network described in [25] (see also the introduction for more details). Even though the sequential model can already achieve very high accuracy, the Inception network performs even better with fewer parameters and much less training data. The learning curve is given in figure 6: it does not show signs of overfitting and clearly demonstrates the quick convergence to almost 100% accuracy.

As presented in figure 5, the network performs equally well over the entire range of $h^{1,1}$ both in the validation and test sets: the variance of the difference between the observed values of the Hodge number and its predictions (i.e. the residuals) is constant as shown by the scatter plot. Moreover, the histogram of the residuals shows that the distribution is peaked around 0 and very few predictions lie far from the central value: the variance is in fact very small.

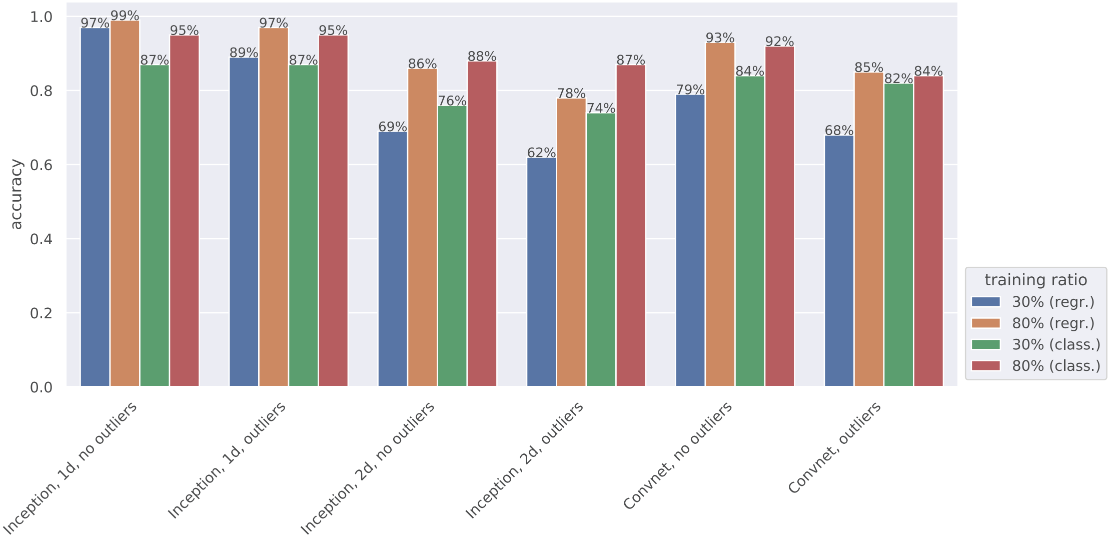

Figure 7. Comparison of accuracy for different properties. The label 'no outliers' means that outliers are excluded from the training and validation set. The labels 1*d* and 2*d* refer respectively to the kernels $(12 \times 1, 1 \times 15)$ and $(3 \times 3, 5 \times 5)$.

3.4. Ablation study

We can now study in detail the relative impact of each improvement introduced in our paper. The three points of comparison are (a) parallel vs sequential convolution layers, (b) using 1*d* kernels 12×1 and 1×15 or 2*d* kernels 3×3 and 5×5 (without changing the number of layers), and (c) including or removing outliers from the training data. A comparison of the accuracy achieved by different models is displayed in figure 7.

First we want to measure the benefit of using parallel instead of sequential convolutions. In [25] we have built a convolutional network (*convnet* in figure 7) made of four layers with 180, 100, 40 and 20 units, all with a 5×5 kernel and ℓ_1 and ℓ_2 regularisation 10^{-4} and 10^{-3} ($\approx$580 000 parameters). The accuracies of this network at a few training ratios are given in table 1 and we refer the reader to [25] for more details. While this network performs better than earlier models (compare figures 1 and 7), its accuracy is below the Inception model.

Second we wish to uncover the effect of using 1*d* kernels 12×1 and 1×15 instead of 2*d* kernels. For this, we have trained a new version of the Inception model with the 1*d* kernels replaced by concurrent 3×3 and 5×5 kernels (typical in computer vision tasks), leaving all other hyperparameters identical ($\approx$290 000 parameters). From figure 7, we find that this network performs even less well than the sequential convolutional network. One possible explanation is that the two 1*d* convolutional windows process separately the information of each single projective spaces (columns) or polynomial equation (rows), scanning all of them one after the other. This could explain why it is necessary to have two 1*d* kernels: one for the projective spaces, one for the equations.

Third we have argued that removing outliers from the training and validation sets helps the network to learn better. The effect is not as important as the previous two points, but still noticeable (figure 7).

Finally we compared the difference between regression and classification. We have one-hot encoded the Hodge numbers, replaced the last layer of the network described in subsection 3.1 by a softmax, and used the cross-entropy loss for optimisation. We find that classification is less efficient than regression (figure 7). Adding two additional Inception modules brings the accuracy to 96%, still below the result from the regression network.

In conclusion we see that convolutional layers working in parallel are responsible for a large part of the performance boost. That convolution is useful for CICY may seem counter-intuitive [13] since the configuration matrices are not rotation nor translation invariant but only permutation invariant. However we first note that convolution alone is only equivariant to global translation: it is not invariant to rotation nor translation (even locally), both of which require the addition of pooling layers (which we do not have) [1]. Moreover convolution layers can be understood more generally as a way to spot different patterns in data by sharing weights, storing them in multiple channels, and recombining them in more complicated representations in subsequent layers. For instance the original Inception models [21–23] include layers with 1×1 kernel, which clearly do not exploit invariance properties. Another motivation for using convolution layers is parameter sharing: the same operations are applied at different locations of the input. Parameter sharing with the 1d shape of the kernels implies that the same formulas are applied to each equation and each projective space, as can be expected for a geometric object.

4. Conclusion

We have introduced a new type of neural network to compute the Hodge number $h^{1,1}$ of complete intersection Calabi–Yau 3-folds. This neural network inspired by Google's Inception model gets near-perfect accuracy, using fewer data and parameters than existing models. This improves largely the prediction power of the network and proves that deep learning is perfectly adapted for computations in algebraic topology. Hence this network should definitely be explored at length to exploit its potential, which seems to be as promising for theoretical physics and mathematics as it has been in computer vision.

The next step consists in predicting also the Hodge number $h^{2,1}$. A preliminary analysis shows that the task is harder and the Inception network reaches only 50% accuracy—but it is higher than all other models, the best of which reach at most 35% (for SVM with Gaussian kernel and sequential convolutional network) [25]. One solution is to use a better representation of the data. A first possibility is to use the favourable representation from [11], but this does not help [25]. Another more promising avenue is to use the graph representation introduced in [16]. It will also be interesting to extend our analysis to other topological objects useful for string theory. A last open question is to understand what the neural network learned and if it is possible to extract any interesting information from the weights. We leave these questions for the future.

Data availability statement

No new data were created or analysed in this study.

Acknowledgments

We are grateful to Sven Krippendorf and Fabian Rühle for discussions, and to Siavash Golkar for comments on the draft. The work of H E has been conducted under a Carl Friedrich von Siemens Research Fellowship of the Alexander von Humboldt Foundation for postdoctoral researchers during part of this project. H E and R F are partially supported by the Miur Prin Contract 2015Mp2cx4 'Non-perturbative Aspects of Gauge Theories and Strings'.

ORCID iDs

H Erbin https://orcid.org/0000-0002-9096-0659
R Finotello https://orcid.org/0000-0002-8472-9004

References

[1] Goodfellow I, Bengio Y and Courville A 2016 *Deep Learning* (Cambridge, MA: MIT Press)
[2] Rosenblatt F 1958 The perceptron: a probabilistic model for information storage and organization in the brain *Psychol. Rev.* 65 6
[3] Lecun Y, Bottou L, Bengio Y and Haffner P 1998 Gradient-based learning applied to document recognition *Proc. IEEE* **86** 2278–324
[4] Bronstein M M *et al* 2017 Geometric deep learning: going beyond Euclidean data *IEEE Signal Process. Mag.* **34** 18–42
[5] Geometric Deep Learning 2021 (available at: http://geometricdeeplearning.com/)
[6] Lei N *et al* 2020 A geometric understanding of deep learning *Engineering* **6** 3
[7] Green P and Hübsch T 1987 Calabi–Yau manifolds as complete intersections in products of complex projective spaces *Commun. Math. Phys.* **109** 99
[8] Ibáñez L E and Uranga A M 2012 *String Theory and Particle Physics: An Introduction to String Phenomenology* (Cambridge: Cambridge University Press)
[9] Candelas P, Dale A M, Lütken C A and Schimmrigk R 1989 Complete intersection Calabi–Yau manifolds *Nucl. Phys.* B **298** 493–525
[10] Green P S, Hübsch T and Lütken C A 1989 All the Hodge numbers for all Calabi–Yau complete intersections *Class. Quantum Grav.* **6** 105–24
[11] Anderson L B, Gao X, Gray J and Lee S-J 2017 Fibrations in CICY threefolds *J. High Energy Phys.* **10** 077
[12] He Y-H 2017 Machine-learning the string landscape *Phys. Lett.* B **774** 564–8
[13] Bull K, He Y-H, Jejjala V and Mishra C 2018 Machine learning CICY threefolds *Phys. Lett.* B **785** 65–72
[14] Bull K, He Y-H, Jejjala V and Mishra C 2019 Getting CICY high *Phys. Lett.* B **795** 700–6
[15] He Y-H and Lee S-J 2019 Distinguishing elliptic fibrations with AI *Phys. Lett.* B **798** 134889
[16] Krippendorf S and Syvaeri M 2020 Detecting symmetries with neural networks (arXiv:2003.13679)
[17] Ruehle F 2017 Evolving neural networks with genetic algorithms to study the string landscape *J. High Energy Phys.* **08** 038
[18] Klaewer D and Schlechter L 2019 Machine learning line bundle cohomologies of hypersurfaces in toric varieties *Phys. Lett.* B **789** 438–43
[19] Brodie C R, Constantin A, Deen R and Lukas A 2020 Machine learning line bundle cohomology *Fortsch. Phys.* **68** 1900087
[20] Ruehle F 2020 Data science applications to string theory *Phys. Rep.* **839** 1–117

[21] Szegedy C *et al* 2015 Going deeper with convolutions *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (https://doi.org/10.1109/CVPR.2015.7298594) (arXiv:1409.4842)

[22] Szegedy C *et al* 2016 Rethinking the inception architecture for computer vision *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* pp 2818–26

[23] Szegedy C, Ioffe S, Vanhoucke V and Alemi A A 2017 Inception-v4, inception-resnet and the impact of residual connections on learning *31st AAAI Conf. Artificial Intelligence*

[24] He K, Zhang X, Ren S and Sun J 2016 Deep residual learning for image recognition *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* pp 770–8

[25] Erbin H and Finotello R 2020 Machine learning for complete intersection Calabi–Yau manifolds: a methodological study (arXiv:2007.15706)

[26] Pedregosa F *et al* 2011 Scikit-learn: machine learning in python *J. Mach. Learn. Res.* **12** 2825–30

[27] Abadi M *et al* 2015 TensorFlow: large-scale machine learning on heterogeneous systems (available at: tensorflow.org)

[28] Chollet F 2015 Keras (https://github.com/fchollet/keras)

[29] Virtanen P *et al* 2020 SciPy 1.0: fundamental algorithms for scientific computing in Python *Nat. Methods* **17** 261–72

[30] Green P S, Hübsch T and Lütken C A 1989 All Hodge numbers of all complete intersection Calabi–Yau manifolds *Class. Quantum Grav.* **6** 105–24

[31] Green P S and Hübsch T 1987 Polynomial deformations and cohomology of Calabi–Yau manifolds *Commun. Math. Phys.* **113** 505

[32] Anderson L B, Gao X, Gray J and Lee S-J 2017 Fibrations in CICY threefolds *J. High Energy Phys.* **10** 077

[33] Anderson L B and Karkheiran M 2018 TASI lectures on geometric tools for string compactifications *Proc. Sci. TASI2017* p 13 (arXiv:1804.08792)

[34] Carifio J, Halverson J, Krioukov D and Nelson B D 2017 Machine learning in the string landscape *J. High Energy Phys.* **09** 157

[35] Kingma D P and Ba J 2014 Adam: a method for stochastic optimization (arXiv:1412.6980)