Inferring Convolutional Neural Networks' accuracies from their architectural characterizations

1st Duc Hoang
Department of Physics
Rhodes College
Memphis, Tennessee 38112 USA
hoadm-21@rhodes.edu

2nd Jesse Hamer
Department of Mathematics
The University of Iowa
Iowa City, Iowa, 52242, USA
jhamer90811@gmail.com

3rd Gabriel N. Perdue
Quantum Science
Fermi National Accelerator Laboratory (FNAL)
Batavia, Illinois, 60510, USA
perdue@fnal.gov

4th Steven R. Young
Oak Ridge National Laboratory
Oak Ridge, Tennessee, 37830, USA
youngsr@ornl.gov

5th Jonathan Miller
Universidad Técnica Federico Santa María
Valparaíso, Chile
jonathan.miller@usm.cl

6th Anushree Ghosh
Universidad Técnica Federico Santa María
Valparaíso, Chile
anushree.ghosh@usm.cl

***Abstract*—The challenge of choosing an appropriate convolutional neural network (CNN) architecture for specific applications and different data sets is still poorly understood in the literature. This is problematic, since CNNs have shown strong promise for analyzing scientific data from many domains including particle imaging detectors. In this paper, we proposed a systematic language that is useful for comparison between different CNN's architectures before training time. This helps us predict whether a network can perform better than a certain threshold accuracy before training up to 70% accuracy using simple machine learning models. Additionally, we found a coefficient of determination of 0.966 for an Ordinary Least Squares model in a regression task to predict accuracy of a large population of networks.**

***Index Terms*—Convolutional neural networks, network architecture, transfer domains, computer vision, high energy physics.**

I. Introduction

Deep Convolutional Neural Networks (CNNs) are a state-of-the-art technique in the fields of computer vision, natural language processing, and other scientific research domains such as High Energy Physics [1], [2]. That said, due to CNNs' inability to generalize for all datasets, a necessary step before applying CNNs to new data is selecting an appropriate set of architecture hyper-parameters. Generally, while there have been many studies of automated architecture search [3], very little has been done to develop a standardized language for describing neural network architectures in such a way as to be useful for comparison of multiple networks, or prediction of network performance metrics on the basis of architectural parameters. In this study, we will thus propose a systematic language to characterize CNN architecture for simple, modular networks (Section II), and focus on demonstrating that different characterizations of the network architecture can be predictive of its performance in two computer vision problems in a particle physics context–vertex finding [2], [4], [5] and hadron multiplicity in MINERvA (Section III). MINERvA [6] is a neutrino-nucleus scattering experiment at Fermi National Accelerator Laboratory with fine-grained, stereoscopic imaging capabilities and few-nanosecond timing resolution. We conclude that our architectural attributes set can be used to give us partial insights into a network's performance prior to training. We will also present specific architectural attributes that are highly relevant to CNNs' performance for those problems for further study and development of the models (Section IV & V).

The networks analyzed here are convolutional networks trained by an evolutionary algorithm called MENNDL (Multi-node Evolutionary Neural Networks for Deep Learning) [7]. The networks were trained for the task of vertex finding [2] and hadron multiplicity counting in images collected from Fermilab's MINERvA detector[1]. In the vertex finding task, for each input image, the location of the point of interaction between incoming neutrino and the target, in terms of which plane in the detector, is the desired output. For the hadron multiplicity problem, we count the number of out-coming charged hadron tracks with sufficient energy to traverse about two planes of the detector from the interaction. The networks were trained using data simulated by state-of-the-art physical models. In order that the networks are insensitive to differences between simulated and real images, some of the network populations were trained with a *domain adversarial* component (DANN) [5], [8]. For this work, we studied two separate output populations of vertex-finding networks and one population of hadron-multiplicity networks, each of which is based on 4,999 repetitions of the evolutionary algorithm. In its running process, only the architecture was varied. The data set of networks analyzed was thus built on a total of 299,050 unique network architectures. We use all networks from the evolutionary process and not only the final networks.

All studies in this paper are reproducible using our analysis, extraction codes, and attributes data set, which are publicly available[2].

[1] minerva.fnal.gov

[2] https://github.com/Duchstf/CNN-Architectural-Analysis

II. Extracted Attributes Summary

Here we describe various network attributes which may be extracted and represented in a uniform way using a minimal amount of computation. Several such attributes are the result of averaging over some groups of attributes. This is because the size of groups of attributes may depend on the specific network architecture, and may not always serve the same functionality or be at the same scale in different networks and thus may produce ambiguity in interpretation. For example, it is tempting to use network depth as an attribute, but different networks might have several input layers or several output layers. To remedy this issue, we can ask for the average depth. Below is a list of all attributes extracted here. The abbreviations used in the analysis are given in [square brackets].

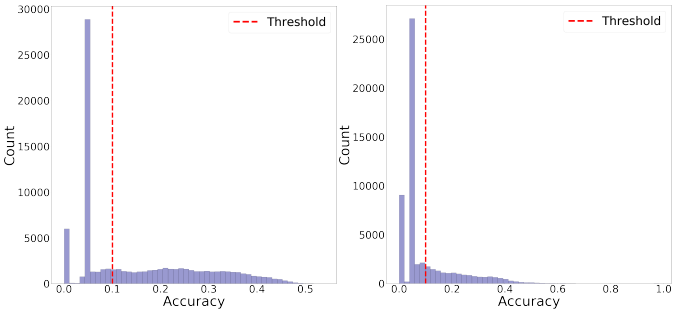
1) Average depth [net_depth_avg]
2) Number of convolutional layers [num_conv_layers]
3) Number of pooling layers [num_pooling_layers]
4) Average number of number of elements in outputs of fully-connected layers [avg_IP_neurons]
5) Average number of connection parameters of fully-connected layers to previous layer [avg_IP_weights]
6) Average number of output feature maps in convolutional layers [num_conv_features]
7) Proportion of convolutional layers followed by a pooling layer [prop_conv_into_pool]
8) Proportion of pooling layers followed by a pooling layer [prop_pool_into_pool]
9) Proportion of convolutional layers with 1×1 kernels [prop_1x1_kernels]
10) Proportion of convolutional layers with square kernel-shapes [prop_square_kernels]
11) Proportion of convolutional layers with horizontally-oriented kernels [prop_horiz_kernels]
12) Proportion of convolutional layers with vertically-oriented kernels [prop_vert_kernels]
13) Number of rectified linear unit (ReLU) activated convolutional layers [num_relu]
14) Number of sigmoid-activated convolutional layers [num_sigmoid]
15) Average percent reduction in activation grid area/ height/ width between consecutive convolutional layers [avg_grid_reduction_area/height/width_consecutive]
16) Average percent reduction in activation grid area/ height/ width between input layers and final convolutional layers [avg_grid_reduction_area/height/width_total]
17) Proportion of convolutional layers using non-overlapping stride [prop_nonoverlapping]
18) Average convolutional stride height/width [avg_stride_h/w]
19) Average ratio of convolutional layer's output feature maps to its depth [avg_ratio_features_to_depth]
20) Average ratio of layer's output feature maps to kernel area/height/width of convolutional layers [avg_ratio_features_to_kerArea/Height/Width]
21) Average ratio of kernel area/height/width to depth of convolutional layers [avg_ratio_kerArea/Height/Width_to_depth]

III. Predict CNNs' performance before training time.

A. Predictions for Vertex-Finding networks

1) Data summary: We analyzed two populations of output networks designed for Vertex Finding in MINERvA using MENNDL. For convenience, we refer to them as the *First* and *Second Populations*. In terms of accuracy, the networks have either 173 or 174 output classes corresponding to planes and targets in the detector. Therefore, the benchmark for random guessing is around 0.6%. In Fig. 1, both populations' accuracy distributions are heavily left skewed with many networks' accuracy clustering around a very low value. Thus, for each population, we split the data into *broken* and *healthy* networks using threshold of 10.05%, which is much higher than random guessing. The threshold was set so that the high peaks of very low performance network in the distributions are included in the *broken* class, and the two classes are balanced. The overall percentage of each category in each population is summarized in Table I.

In this task, we choose to not combine the two populations together for fear that the mentioned inherent difference in the networks' attributes can interfere with our classification task and cause difficulties in interpreting the results. For regression, we chose to combine the two populations together on the basis that we are only looking at the correlations of the network's attributes to predict the accuracy.



(a) (b)

Fig. 1. Classification accuracy distributions for each population for vertex finding. Left: accuracy distribution of the first population. Right: accuracy distribution of the second population. The dotted lines represents where each dataset was divided into "broken" or "healthy" for our classification task.

TABLE I
Fraction of broken and healthy networks in each population.

Population	*Broken*	*Healthy*	Total number of networks
First	50.0%	50.0%	83966
Second	50.0%	50.0%	31542

2) Classification results: Each population dataset was randomly split into training and testing sets with a 80/20 ratio, respectively. Here we used Random Forest (RF) [9] and Extremely Randomized Tree (ERT) [10] to perform classification. For this task, we propose a base accuracy of 50%, since there is no class imbalance in both populations we used for classification. The primary purpose of building machine learning models was to demonstrate the predictive nature of the architectural attributes, but not to perform further analysis based on the outputs of the models. As can be seen in Table II, the scores are significantly better than random guessing (50%), which underlines that the models were able to detect architectural separation between the attribute sets for *broken* and *healthy* networks. Furthermore, the cross-validation scores and the accuracy on test set are very close together, so we would expect the models to have the same accuracy on unseen data set.

3) Regression results: After performing healthy/broken classification, we performed regression on just the healthy networks. Since the broken networks' accuracy distribution is heavily left-skewed, we couldn't perform regression on them. Before fitting, interaction terms between the original attribute set are also added. Using a non-linear Ordinary Least Square (OLS) model with linear parameters, we performed regression separately on each population and then combine them together.

The results from the fit are summarized in Table III. A general trend is that as the number of networks increase, the R^2 value gets better. This suggests that while we don't have enough events in the sub-populations to get a good fit, they overlap enough in the right regions of phase space to allow a good fit altogether. However, it is worth noting that, as depicted in Fig. 2, while the majority of residuals are distributed around 0, there seems to be a linear relationship between the residual and the fitted values, which means that more regressors are needed to account for this behaviour. Furthermore, the Quantile-Quantile (Q-Q) plot in Fig. 2 with a high right tail indicates that there is a gap in the distribution of the residuals. This is due to the fact that the accuracy's distribution is heavily left skewed with very few networks with high accuracy. Note that we also tried several regression algorithms that can account for a high level of non-linearity in the data. Almost all of them fail to generalize to validation data set and do not provide a significantly better R^2 than a simple OLS model.

TABLE II
ACCURACY OF RF AND ERT ON TRAIN SET AND VALIDATION SET IN FIRST & SECOND POPULATION OF VERTEX-FINDING NETWORKS.

Models	Population	Average accuracy scores	
		Cross-validation	On test set
RF	First	67.3 ± 0.004%	66.8%
	Second	69.6 ± 0.006%	70.7%
ERT	First	66.7 ± 0.006%	66.0%
	Second	69.6 ± 0.007%	70.3%

TABLE III
R^2 VALUE OF NON-LINEAR OLS MODEL ON INDIVIDUAL POPULATIONS AND COMBINED.

Population	R^2	Adjusted R^2	Number of healthy networks
First	0.445	0.439	41984
Second	0.298	0.275	15771
Combined	0.966	0.966	57755

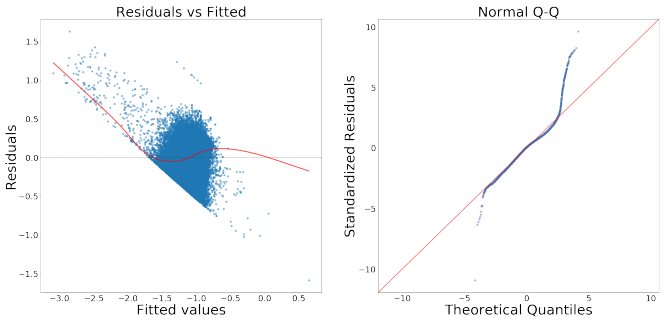


Fig. 2. Residuals analysis of OLS model on the combined data set of healthy networks. Left: Scatter plot between residuals and the fitted value, indicating a linear relatioship between residuals and fitted. Right: Quantile-quantile plot depicting the distribution of standardized residuals – the high tail indicates a gap in the residuals distribution.

B. Prediction for Hadron-Multiplicity networks

Similar to previous section's analysis for vertex-finding networks. To prevent class imbalances in the training data, we set the threshold to be 0.38 and broken networks were randomly sampled so that we have a 50/50 distribution between the two classes of 34614 networks in total. For this task, we again used RF and ERT models to classify between broken and healthy networks. The classification results are reported in Table IV. Both models consistently achieve accuracy of more than 70% in both cross-validation on training set and testing set, which is 20% better than random guessing (50%), since there is no class imbalance. Here we do not present regression's results for hadron multiplicity networks, since we have such a small amount of networks that the regression results are not significant to be presented.

IV. ATTRIBUTE ANALYSIS

Here we give some examples of how the attributes set can potentially be used to analyze the behaviour of the network's architecture. After fitting the OLS model, we found many attributes that are more significant and have much larger coefficients than other attributes. They are reported in Table V.

As can be seen, `net_depth_avg`, `avg_IP_neurons` and their interactions are strongly correlated with the perfor-

TABLE IV
ACCURACY OF RF AND ERT ON TRAIN SET AND VALIDATION SET.

Model	Average cross-validation score	Accuracy on test set
RF	70.3 ± 0.006%	70.6%
ERT	70.2 ± 0.003%	70.5%

TABLE V
ATTRIBUTES THAT HAVE SIGNIFICANTLY LARGER COEFFICIENTS THAN THOSE OF OTHER ATTRIBUTES IN OLS MODEL.

Variable	Coefficient
net_depth_avg	3.5 ± 0.03
avg_IP_neurons	2.8 ± 0.02
avg_IP_neurons*net_depth_avg	1.6 ± 0.01
avg_grid_reduction_height_total*avg_stride_h	-0.5 ± 0.02
avg_IP_neurons*num_conv_layers	-0.9 ± 0.01
avg_IP_neurons*num_pooling_layers	-1.1 ± 0.01
num_conv_layers	-2.0 ± 0.02
num_pooling_layers	-2.5 ± 0.02

mance. This suggests that increasing the capacity (number of parameters) of fully connected layers in the CNN can improve the overall performance of the CNN model. Additionally, `num_pooling_layers` and `num_conv_layers` are negatively correlated with the performance. This implies that, as we add more convolutional layers and pooling layers into the model, its performance will generally decrease. While the rest of the interactions are harder to interpret, the interaction term between `avg_grid_reduction_height_total` and `avg_stride_h` seems to point out an interesting property. Typically in computer vision problems only square kernels are ever considered. MINERvA physicists studied asymmetric kernel shapes for the vertex finding problem as a way of keeping the convolutions from reducing the image size along the neutrino direction axis [4], [5].

Thus, analyzing the important features of the machine learning models can give us insights into how to potentially improve a CNN model's performance.

V. SUMMARY AND OUTLOOK

In this paper, we proposed a systematic method that can be useful for uniform comparison of different architectural attributes of CNNs. We demonstrated the predictive nature of those attributes in two specific problems—vertex finding and hadron multiplicity counting in MINERvA—through building machine learning models that predict the CNN's performance before its training time.

For future work, we plan to extend the architectural attributes set and take into account other hyper-parameters related to input domains and training process, which might provide us with a more comprehensive study of network performance. Furthermore, it can be interesting for us to perform the same analysis on state-of-the-art network architectures and see to what extent does our current set of architectural attributes correctly characterize the network's performance. It is also promising to incorporate machine learning models such as the ones we built in this paper into model selection algorithms to evaluate a network's accuracy before training time, thereby boosting the efficiency of the algorithms.

ACKNOWLEDGMENT

We would like to thank the MINERvA collaboration for access to their simulated data sets for this analysis. MINERvA uses the resources of the Fermi National Accelerator Laboratory (Fermilab), a U.S. Department of Energy, Office of Science, HEP User Facility. Fermilab is managed by Fermi Research Alliance, LLC (FRA), acting under Contract No. DE-AC02-07CH11359, which included the MINERvA construction project. This material is based upon work supported by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research, Robinson Pino, program manager, under contract number DE-AC05-00OR22725. This research used resources of the Oak Ridge Leadership Computing Facility at the Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725.

REFERENCES

[1] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, pp. 436 EP –, 05 2015. [Online]. Available: https://doi.org/10.1038/nature14539

[2] L. Song, F. Chen, S. R. Young, C. D. Schuman, G. N. Perdue, and T. E. Potok, "Deep learning for vertex reconstruction of neutrino-nucleus interaction events with combined energy and time data," *CoRR*, vol. abs/1902.00743, 2019. [Online]. Available: http://arxiv.org/abs/1902.00743

[3] M. Wistuba, A. Rawat, and T. Pedapati, "A survey on neural architecture search," *CoRR*, vol. abs/1905.01392, 2019. [Online]. Available: http://arxiv.org/abs/1905.01392

[4] A. M. Terwilliger, G. N. Perdue, D. Isele, R. M. Patton, and S. R. Young, "Vertex reconstruction of neutrino interactions using deep learning," in *2017 International Joint Conference on Neural Networks (IJCNN)*, May 2017, pp. 2275–2281.

[5] G. Perdue *et al.*, "Reducing model bias in a deep learning classifier using domain adversarial neural networks in the MINERvA experiment," *Journal of Instrumentation*, vol. 13, no. 11, pp. P11 020–P11 020, nov 2018. [Online]. Available: https://doi.org/10.1088%2F1748-0221%2F13%2F11%2Fp11020

[6] L. Aliaga *et al.*, "Design, calibration, and performance of the MINERvA detector," *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, vol. 743, pp. 130 – 159, 2014. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0168900214000035

[7] S. R. Young, D. C. Rose, T. P. Karnowski, S.-H. Lim, and R. M. Patton, "Optimizing deep learning hyper-parameters through an evolutionary algorithm," in *Proceedings of the Workshop on Machine Learning in High-Performance Computing Environments*, ser. MLHPC '15. New York, NY, USA: ACM, 2015, pp. 4:1–4:5. [Online]. Available: http://doi.acm.org/10.1145/2834892.2834896

[8] Y. Ganin, E. Ustinova, H. Ajakan, P. Germain, H. Larochelle, F. Laviolette, M. March, and V. Lempitsky, "Domain-adversarial training of neural networks," *Journal of Machine Learning Research*, vol. 17, no. 59, pp. 1–35, 2016. [Online]. Available: http://jmlr.org/papers/v17/15-239.html

[9] L. Breiman, "Random forests," *Mach. Learn.*, vol. 45, no. 1, pp. 5–32, Oct. 2001. [Online]. Available: https://doi.org/10.1023/A:1010933404324

[10] P. Geurts, D. Ernst, and L. Wehenkel, "Extremely randomized trees," *Machine Learning*, vol. 63, no. 1, pp. 3–42, Apr 2006. [Online]. Available: https://doi.org/10.1007/s10994-006-6226-1