A Comprehensive Co-Evolutionary Framework for DNN Architecture Design Space Exploration on GPU Accelerator Platforms

1st Given Name Surname
dept. name of organization (of Aff.)
name of organization (of Aff.)
City, Country
email address or ORCID

2nd Given Name Surname
dept. name of organization (of Aff.)
name of organization (of Aff.)
City, Country
email address or ORCID

3rd Given Name Surname
dept. name of organization (of Aff.)
name of organization (of Aff.)
City, Country
email address or ORCID

4th Given Name Surname
dept. name of organization (of Aff.)
name of organization (of Aff.)
City, Country
email address or ORCID

5th Given Name Surname
dept. name of organization (of Aff.)
name of organization (of Aff.)
City, Country
email address or ORCID

6th Given Name Surname
dept. name of organization (of Aff.)
name of organization (of Aff.)
City, Country
email address or ORCID

***Abstract*—Deep Neural Networks(DNNs) are an extremely attractive subset of computational models which provide promising results for a wide variety of problems. However, the performance delivered by DNNs overshadows the network architecture search(NAS) process and network's suitability for a given task. In this paper, we present an end-to-end framework designed to detect the abstract model required for a given data set, genetically modify the network architecture and evolve the knowledge representation format. The inherent parallelism offered by both the neural network and it's evolutionary extension is exploited by deploying the model on a GPU which substantially improves the throughput of Genetic-NAS.**

***Index Terms*—component, formatting, style, styling, insert**

I. Introduction

Deep Neural Networks (DNNs) are parallel computational structures used for implementation of static and dynamic systems through a process of understanding the problem stated rather than memorizing the input-output relation. The steady rise in DNNs popularity can be attributed to the influx of new architectures such as AlexNet [1], ResNet [2], VGG [3] and LSTM [4] among others along with a surge in hardware specifically designed for parallel computation such as GPUs, ASICs and FPGAs. Moreover, the algorithms governing the functionalities of the different neural networks have a strong underlying mathematical foundation which has only reinforced over the years with Improved Back-Propogation [5] and Conjugate Gradient Algorithm [6] being prime examples. The performance (P) of a DNN can be expressed as a function of three main factors: network architecture (A), network parameter set (NP) and knowledge representation of the data provided (K).

$$P = f(A, N_P, K) \tag{1}$$

Much empahsis has been placed on developing new algorithms and programming paradigms for modifying the network parameters to improve the performance. This overwhelming infatuation has often undermined the importance of network architecture selection and its association with the problem statement provided. Given no prior knowledge, the network architecture search (NAS) process has been a trivial trail and error process which is painstakingly human. Moreover, given a network architecture and an optimized parameter set, the configuration stands good only for a given dataset and does not account for an online model with wild flucuations in the admitted dataset. Modern NAS techniques can be subdivided into Reinforcement Learning based, such as [7] and [8], Evolutionary Techniques, such as [9], and Gradient based Techniques, such as [10]. RL techniques suffer a high computational cost for combing through the entire search space and adapting to the environment, which Gradient based techniques are not adaptive in nature and hence cannot be fit onto an online model.

Applying a random search on network architectures, would be time consuming and computationally expensive as the total number of configurations grows exponentially with even minor changes to the architecture. Recent advances in hardware has increased the speed of applications with inherent parallelism such as DNNs and Genetic Programming (GP) as multiple threads of execution cooperate to complete the work in a shorter time frame. Programming models like MPI and CUDA were developed to translate optimized data transfer operations and its overlap with I/O operations from a logical view point to an implementation format.

In this paper we present a dynamic end-to-end framework composed of intelligent subsystems, each one tackling one aspect of a network's performance stated in Equation 1 to

overcome the aforementioned shortcomings. The key features of our framework are,

- Data Aware Model Selection: A robust subsystem accepts the data either in the form of a single file or a directory and analyses it, based on various signals such as temporal difference, spatial difference, dimensionality analysis and data format. The queued data is assigned flags based on the signal results obtained. The data is parsed through an optimized scheduler which seeks to limit the time difference between successive data admissions which leads to increased throughput.
- Pattern Aware Pruning: After selecting a base model for the task, genetic programming is introduced to grow out the model to its optimal structure. In order to limit the growth of the weight matrix, pruning was applied to the genetic tree so formed. The pruning areas were determined by applying Markov chain analysis to the search pattern and the corresponding pobabilistic states.
- Layer-wise Competition: For each layer, multiple instances were created and the fitness of each instance is evaluated after every iteration. The NAS process was shifted away from an accuracy-driven model and more emphasis was given to individual contribution of each instance. Back-propogation algorithm was used to fine tune the model and create a environment where multiple instances of the same layer are forced to compete against each other.

II. Related Work

Recent efforts of integrating multi-disciplinary algorithms with NAS has given rise to a copius amount of techniques to automate the search process. As mentioned previously, the efforts can be broadly classified as Reinforcement Learning based, Evolutionary algorithm based and Gradient based techniques. Reinforcement Learning based techniques treats the entire search space as a learning environment with parameter matrix of each layer's instance having either a positive or negative feedback. An external agent is employed to roam the search space and learn the environment based on a pre-decided reward system. Evolutionary algorithms can be further subclassified into Evolutionary programming and Genetic Programming. Both paradigms attempt to emulate Drawin theory of evolution from different approaches with the help of genetic operators such as selection, crossover and mutation. Gradient based techniques utilize the gradient descent algorithm to test the goodness-of-fit of each applicable layer parameter.

- Reinforcement Learning based: In [11] the authors generate high-performing CNN architectures for a given learning task using Q-learning with an ϵ-greedy exploration strategy and experience replay; In [12] the authors formulate the NAS as a Markov decision process which enables a more effective RL-based search. The samples generated by the previous policies are used efficiently for an off-policy GAN search algorithm.
- Evolutionary based: In [13] the authors construct a simplified supernet where all architectures are single paths which solves the weight co-adaption problem. Conduction of the search process in complex environments under multiple constraints is cnoducted seamlessly through uniform path sampling. In [14] evolved machine learning algorithms are created from scratch. They apply evolutionary concepts on top of top algorithms such as bilinear interactions, normalized gradients, and weight averaging.
- Gradient based: Using Gradient descent offers a big advantage in terms of reduced GPU-hours as shown by [15]. The entire search space is represented in the form of directed acyclic graphs each one of which constitutes a possible network architecture. A differential sampler is then trained through gradient descent which optimizes the learning process. The resultant network took only 4 GPU hours with a test error of 2.82%. Another method involved optimizing the network configuration through stochastic relaxation as shown by [16]. The authors apply gradient descent technique with adaptive step-size method for low cost computations.
  Techniques to train the weight matrix of a nerual network were dominated by variants of gradient descent algorithms during the early years of DNNs. Slowing genetic operations were introduced into the process of training the DNN. These operations provide a global search with the added advantage of loval minima avoidance. In [17] a GA is used to evolve ecological neural networks that can adapt to their changing environment. This is achieved by letting the fitness function, which in this case is seen as individual for every gene, co-evolve with the weights of the network.
  n [18] and [19], a genetic algorithm is used on a fixed three layer feedforward network to find the optimal mapping from the input to the hidden layer (i.e. the set of optimal hidden targets). In the evaluation phase, the weights from the hidden to the output layer are learned using a simple supervised gradient descent rule. The search space is not the weight space but the hidden target space.

III. Co-Evolutionary Model+Architecture Search

In this section we dicuss the model selection process along with the co-evolutionary architecture search process and the corresponding GPU partioning of the training phase. The goal of developing an end-to-end framework consists of three main components, each one focusing on one aspect of optimizing the performance of a neural network. An additional component in the form of GPU implementation is added to provide a comparative study and to speed up the co-evolutionary NAS training proccess. Note that a GPU-partitioned architecture has fewer weights because not all layers have interconnections. Dropping some of the interconnections reduces the communication time between the processors and therefore helps in efficiency.

A. Base Model Selection

The overview of the model selection process is shown in Figure 1.

A central data repository is created for data admission and for each data entry flags are initialised as null vectors. Depending on the workload provided, each entry is scheduled as a job and sequenced for processing. A non-preemtive approach is applied to the scheduling task for higher throughput and low scheduling overhead. The scheduler automates the tedious part of Machine Learning by intelligently exploring multiple pipelines to find the most apt on for the dataset provided. Once the jobs are optimally sequenced, they pass through a set of filters. These filters provide information about the job in terms of:

- Polynomial features
- Dimensionality
- Temporal dependencies
- Spacial dependencies
- Stacking Estimation

The filtered output has a single feedback pass into the repository where the flags are updated. Based on the flag configuration, a base model is chosen and after a set number of cycles the model is fed into the next subsystem along with the data.

B. Genetic Programming based NAS and weight optimization

Once a base model such as Convolutional Neural Network (CNN) or Feed-Forward Network (FNN) is selected, genetic programming is applied to build upon the base. The general flow of this growth for a CNN is shown in Figure 2.

A complementary co-dependent structure for optimising the NAS process and the weight selection is employed using structured chromosomes in the representational space. For computational purposes, an intermediate state of binary matrix encoding is introduced. An n-dimensional matrix scheme has been implemented to represent the neural network structures in a concise manner. It encodes regularities in the connectivity matrix defining the network structure. Another added advantage of the binary representation is the translation of genetic operators as structural changes in the network which in turn depending on the chromosome layer they are acting on. A many-to-one mapping is employed between the architectural representation and the weight representation to have a two-step verification of the goodness-of-fit of the model as shown in Figure 3.

1) Pruning: In the evaluation stage of the encoding system, the generated networks are first pruned before they are trained using the back propagation module. Pruning removes neurons and the corresponding links that have no incoming or outgoing connections. It does this recursively until all such neurons have been removed. Back propagation training is performed for a predefined number of cycles, rather than the more customary process of stopping at a required error level, since network convergence is a factor of the network's configuration. The optimal number depends on the dataset provided and the train-test split set used. The back propagation module is a standard one using the normal gradient descent weight updating rule with a momentum term. The module uses a per-pattern's weight update mechanism, meaning that the weights are updated after every iteration of the training phase

The fitness function used to evaluate the goodness-of-fit of

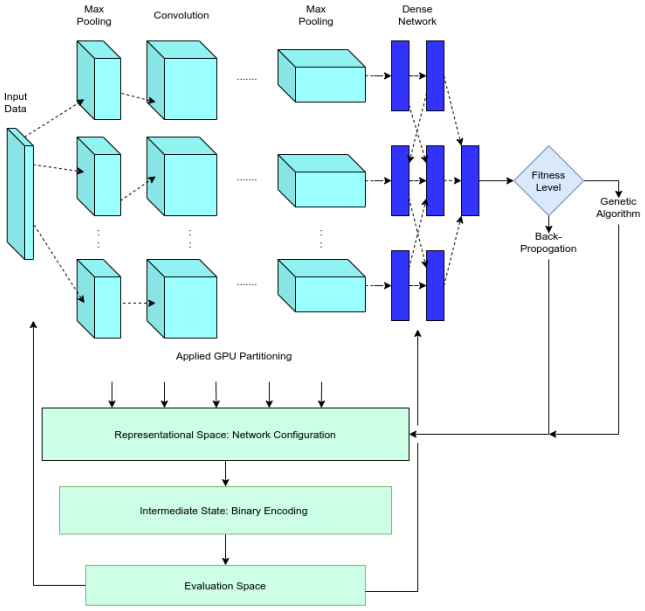

Fig. 2. Genetic Programming based NAS

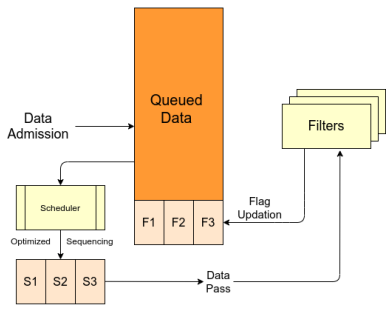

Fig. 1. Base Model Selection

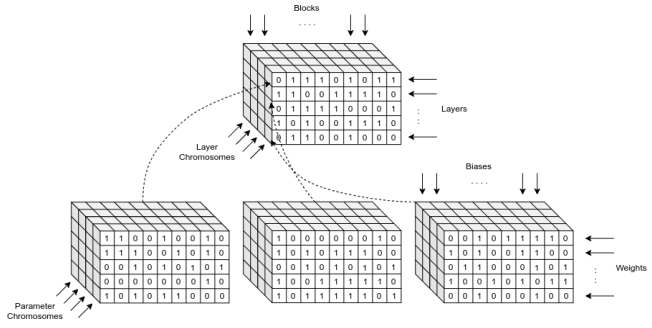

Fig. 3. Many-to-one relation between NAS and weight representation

the network instance is given by Equation 2

$$F = \frac{\hat{Y} - Y}{\sum_{i=0}^{N} O_{\mathrm{i}}} + \alpha \frac{C}{C_{\max}} + \mu P \tag{2}$$

where $\hat{Y}$ is the target variable, Y is the predicted output, O_{i} is the number of output classes, C is the network complexity, $C_{\max}$ is the maximum complexity, P is the fraction of the network pruned, α is the learning rate and β is momentum. Note that since the fitness funciton is directly proportional to the loss, the problem of optimizing the neural network is a minimization problem rather than the usual maximization problem.

2) Markov Chain analysis: The most common methods for analysing genetic structures include Schema theorem, Price's theorem and building block hypothesis. The methods mentioned provide a good approximation of the genetic model and have been used often. Markov Chain analysis, however replicates an exact model of the genetic structure in a state space diagram. The models so formed provide useful insight about the effect genetic operators have on the transition matrix. The transition matrix is the binary encoded intermediatory representation of the network connections as chromosomes.

Since binary encoding was employed, the chromosomes are treated as binary strings with each gene representing "no connection" or 0 and "connection" or 1. If N chromosomes of lenth l are generated, the number of possible states are $N + 1$. State i is referred to as the state with exactly i ones and $(l - i)$ zeros. The operation of the genetic algorithm is now defined by a $(l+1) * (l+1)$ transition matrix $P[i, j]$ that maps the current state i to the next state j . The probability of a transition from state i to state j is given by one entry in the matrix $p(ij)$. A sample markov model for l = 1 and N = 4 is shown in Figure 4.

If f_0 is the fitness of the chromosome of length l having i zeros,when the connection is severed and f_1 is the fitness of the chromosome when the connection is retained the probability of connection surviving is given by Equation 3.

$$p_1 = \frac{i * f_1}{i * f_1 + (l - i) * f_0} \tag{3}$$

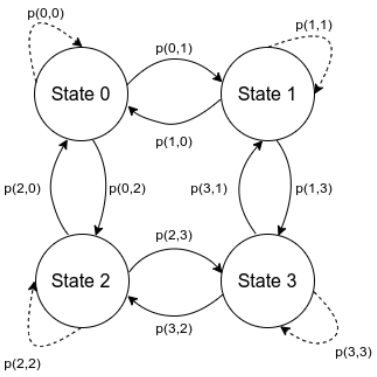

Fig. 4. Markov Analysis

And the probability of connection not surviving is given by Equation 4.

$$p_0 = \frac{(l - i) * f_0}{i * f_1 + (l - i) * f_0} \tag{4}$$

Hence the probability of transition from state i to j can be written as:

$$p(i, j) = \binom{l}{j} (p_1)^{\mathrm{j}} (p_0)^{\mathrm{l\text{-}j}} \tag{5}$$

Equation 5 completely discribes the transition matrix with no preference for a state(i.e. a population) with all-ones or a state with all-zeros, and the equation reduces to the one for pure genetic drift.

IV. Experimental Results

In this section we describe the setup followed for conduction of our experiment and follow it up with a discussion of results and findings of similar works. The genetic programming algorithm has been implemented for the NAS process while evolutionary operators along with back propogation was used for parameter setting and network convergence. The rig setup for our expirement consisted of an Intel(R) Core(TM) Xeon Platinum E5-2682 CPU@2.50GHz with a 500 GB DDR memory. The neural network was developed through Tensorflow v2.3 and deployed on a PowerEdge R740 server with 1 NVIDIA Tesla V100-PCIe GPU card, which has 5120 CUDA cores with a peak throughput of 112 TFLOPS. The experiment was conducted on 6 public datasets: MNIST digit classification [20], CIFAR10, CIFAR 100 [21], Boston Housing price regression dataset [22], IMDB movie review sentiment dataset [23] and Reuters newswire classification dataset [24]. The network settings are described in Table I.

1) Model selection: Initial expermients were conducted on text based datasets i.e, [23], [24] and [22]. The datasets were admitted one by one and the respective flags were initialized to null vectors. The filter set described in the previous section is utilized to search through the possible avenues for our data to traverse through. The dataset is passed through the filter set for 10 cycles which cements the tuning result. Note that the dataset is passed through the filters in a divided format in order to prevent oversaturation of the filter output. The number of divisions in the sampled dataset is equal to the number of cycles. The throughput percentage for text based datasets (Orange) and image based datasets (Blue) is shown in Figure 5.

2) Genetic-NAS: Both the structural part of the chromosomes and the connections use binary encoding. The coding of the structural part is, in general, non-homogeneous in that it consists of different parts each having its own gene length. These parts correspond to the rewriting rules for one or more rewriting cycles. In the present implementation only homogeneous chromosomes were used because implementing non-homogeneous chromosomes in the core genetic programming used is quite difficult.

Algorithm 1: Co-evolutionary NAS with parameter updation

Input: Training Set D, Fitness threshold λ, Base model M, Maximum Network Depth max_depth, Maximum Nodes max_nodes, Maximum Complexity C, Genetic Operators G

Result: Optimal Network Configuration is obtained

Initialize I instances of Base Model with single block;
Start initial run;
for *Each n in* max_nodes **do**
  Develop the network under constraint C and max_depth; **for** *Each instance* i *in* I **do**
    Store fitness scores in array $scores$[];
  end
end
for *Each Instance* i *in* I **do**
  Obtain Network configuration in an intermediate space through binary encoding; Obtain I chromosomes; **for** *Each potential solution chromosome in* I **do**
    Establish a many-to-one relation with w parameter representations;
    Using G evolve the chromosome;
  end
  Update $scores$[];
end
Translate the intermediate state to a binary evaulation tree;
for *Each node* n *in chromosome* i **do**
  Toggle the connection to the next layer using mutation;
  if *Fitness score decreases* **then**
    Apply Back propogation;
    Prune(Connection$(i_n) -> ((i+1)_n)$);
  end
  else
    if *Any chromosome* i *decreases below* λ **then**
      break;
    end
    else
      continue;
    end
  end
end

TABLE I
NETWORK SETTINGS

Parameter	Setting
Learning Rate	X ∈ [0.0001-1.0]
Momentum	X ∈ [0.0-0.9]
Back Propogation Cycles	<50
Encoding	Binary
Selection	Elitism
Mutation	Binary Inversion
Crossover	Single Point
Mutation Rate	X ∈ [0.0-1.0]
Crossover Rate	X ∈ [0.001-1.0]
Maximum Depth	5
Maximum Nodes	1024
l	31
N	10
Number of Generations	10
Number of epochs	10
Replacement	Unconditional

Figures 6 and 7 show the relation between network configuration of the image dataset and text dataset systems and their binary encoded formats. Both systems showed very similar behaviour on this problem. Convergence curves of the best individual in the population vs generation were nearly identical as shown in the Plots 8 and 9.

V. CONCLUSION AND FUTURE WORK

REFERENCES

[1] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. ImageNet Classification with Deep Convolutional Neural Networks. In NIPS.

[2] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In CVPR.

[3] Karen Simonyan and Andrew Zisserman. 2015. Very Deep Convolutional Networks for Large-scale Image Recognition. In ICLR.

[4] Sepp Hochreiter; Jürgen Schmidhuber (1997). "Long short-term memory". Neural Computation.

[5] Nawi N.M., Ransing R.S., Salleh M.N.M., Ghazali R., Hamid N.A. (2010) An Improved Back Propagation Neural Network Algorithm on Classification Problems. In: Zhang Y., Cuzzocrea A., Ma J., Chung K., Arslan T., Song X. (eds) Database Theory and Application, Bio-Science and Bio-Technology. BSBT 2010, DTA 2010. Communications in Computer and Information Science, vol 118. Springer, Berlin, Heidelberg.

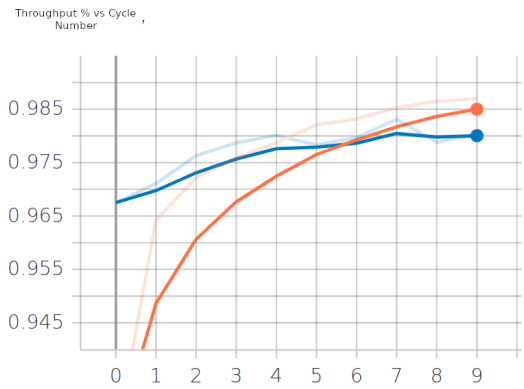

Fig. 5. Throughput % vs Number of Cycles

[6] B. Wang, B. Yang, J. Sheng, M. Chen and G. He, "An Improved Neural Network Algorithm and its Application in Sinter Cost Prediction," 2009 Second International Workshop on Knowledge Discovery and Data Mining, Moscow, 2009, pp. 112-115, doi: 10.1109/WKDD.2009.180.

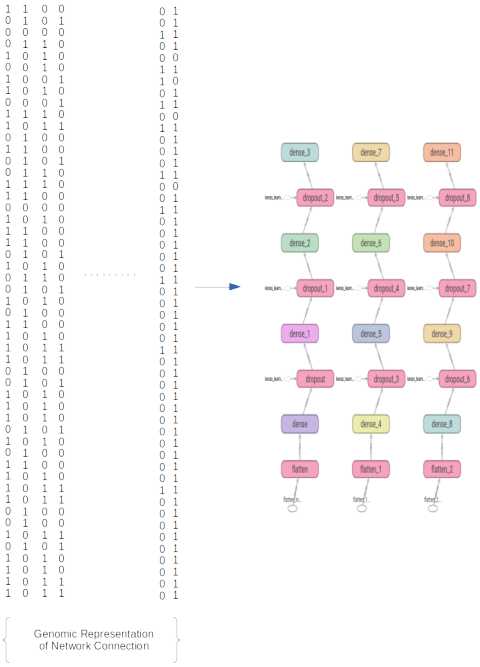

Fig. 6. Boston dataset genomic encoding

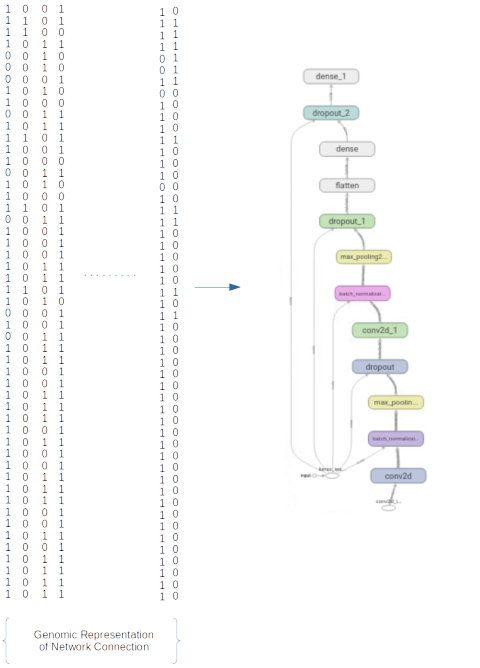

Fig. 7. CIFAR10 dataset genomic encoding

[7] Barret Zoph, Quoc V. Le, Neural Architecture Search with Reinforcement Learning, arXiv:1611.01578
[8] H. Liu and C. Zhang, "Reinforcement Learning based Neural Architecture Search for Audio Tagging," 2020 International Joint Conference on Neural Networks (IJCNN), Glasgow, United Kingdom, 2020, pp. 1-8, doi: 10.1109/IJCNN48605.2020.9207530.
[9] Clifford Broni-Bediako, Yuki Murata, Luiz Henrique Mormille, Masayasu Atsumi, Evolutionary NAS with Gene Expression Programming of Cellular Encoding, arXiv:2005.13110
[10] Liam Li, Mikhail Khodak, Maria-Florina Balcan, Ameet Talwalkar, Geometry-Aware Gradient Algorithms for Neural Architecture Search, arXiv:2004.07802
[11] Baker B, Gupta O, Naik N, et al. Designing neural network architectures using reinforcement learning[J]. arXiv preprint arXiv:1611.02167, 2016
[12] Yuan Tian, Qin Wang, Zhiwu Huang, Wen Li, Dengxin Dai1, Minghao Yang, Jun Wang, and Olga Fink, Off-Policy Reinforcement Learning for Efficientand Effective GAN Architecture Search. arXiv:2007.09180
[13] Zichao Guo, Xiangyu Zhang, Haoyuan Mu, Wen Heng, Zechun Liu, Yichen Wei, Jian Sun, Single Path One-Shot Neural Architecture Search with Uniform Sampling, arXiv:1904.00420
[14] Esteban Real, Chen Liang, David R. So, Quoc V. Le, AutoML-Zero: Evolving Machine Learning Algorithms From Scratch, arXiv:2003.03384
[15] Xuanyi Dong, Yi Yang Searching for A Robust Neural Architecture in Four GPU Hours. arXiv:1910.04465
[16] Youhei Akimoto, Shinichi Shirakawa, Nozomu Yoshinari, Kento Uchida, Shota Saito, Kouhei Nishida, Adaptive Stochastic Natural Gradient Method for One-Shot Neural Architecture Search, arXiv:1905.08537
[17] Lund, H.H. and Parisi, D., "Simulations with an Evolvable Fitness Formula" Technical Report PCIA-1-94, C.N.R., Rome, 1994.
[18] Hassoun, M.H., Fundamentals of Artificial Neural Networks, MIT Press, 1995.
[19] Munro, P.W., "Genetic Search for Optimal Representations in Neural Networks", International Conference on Artificial Neural Nets and Genetic Algorithms (ANNGA93), Innsbruck, Austria, pp. 628-634, 1993.
[20] LeCun, Yann and Cortes, Corinna. "MNIST handwritten digit database." (2010):
[21] Learning Multiple Layers of Features from Tiny Images, Alex Krizhevsky, 2009.
[22] Harrison, D. and Rubinfeld, D.L. Hedonic prices and the demand for clean air, J. Environ. Economics and Management, vol.5, 81-102, 1978.
[23] Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. (2011). Learning Word Vectors for Sentiment Analysis. The 49th Annual Meeting of the Association for Computational Linguistics (ACL 2011).
[24] Sam Dobbins, Mike Topliss, Steve Weinstein Reuters-21578, 1987

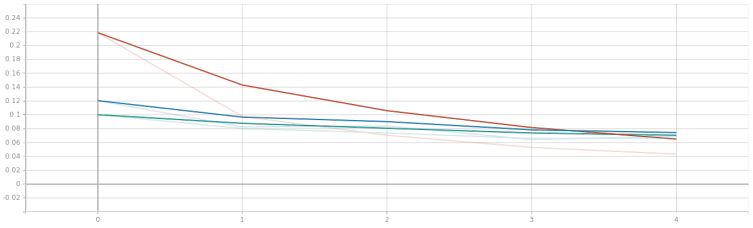

Fig. 8. Loss for Image based Datasets

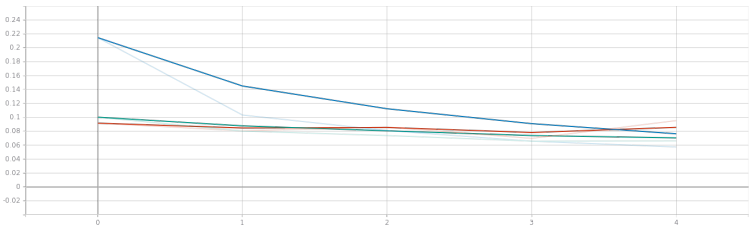

Fig. 9. Loss for Text based Datasets