Sustainable Cities and Society 67 (2021) 102747

Contents lists available at ScienceDirect

Sustainable Cities and Society

journal homepage: www.elsevier.com/locate/scs

Reducing energy consumption of Neural Architecture Search: An inference latency prediction framework

Longfei Lu, Bo Lyu *

School of Computer Science and Engineering, University of Electronic Science and Technology of China, Chengdu 611731, China

ARTICLE INFO

Keywords:
Deep learning
Latency prediction
Neural architecture search
Energy saving
Society and environment

ABSTRACT

Benefit from the success of NAS (Neural Architecture Search) in deep learning, humans are hopefully been released from the tremendous labor of manual tuning of structure and hyper-parameters. However, the success of NAS comes at the cost of much more computational resource consumption, thousands of times more computational power than ordinary training of manual-designed models, especially for the resource-aware multi-objective NAS, which must be serialized as a sequential loop of sampling, training, deployment, and inference. Recent research has shown that deep learning leads to huge energy consumption and CO2 emission (training of the namely Transformer can emit CO2 as much as five cars in their lifetimes Strubell et al. (2019)). Aiming to alleviate this issue, we propose the end-to-end inference latency prediction framework to empower the NAS process with a direct resource-aware efficiency indicator. Namely, we first propose the end-to-end latency prediction framework, which can predict latency quickly and accurately based on the dataset collected by ourselves. Eventually, we experimentally show that with the encoding scheme we designed, our proposed best model, LSTM-GBDT Latency Predictor(LGLP) achieves an excellent result of **0.9349 MSE, 0.5249 MAE, 0.9842 R^2, and 0.9925 corrcoef**. In other words, our limited dataset and encoding scheme already provide the precise knowledge representation of this large search space. By equipping NAS with the proposed framework, taking NEMO for example, it will save **1588 kWh·PUE energy, 1515 pounds CO2 emissions, and $3176 cloud compute cost of AWS**. For NAS is now widely exploited in research or industry applications, this will bring incalculable benefits to society and the environment.

1. Introduction

The sustainable development of the environment and city is attracting more and more attention from science and technology circles. Ignoring the influence of social environment and blindly pursuing economy is not a long-term healthy development path (Owyer, Pan, Charlesworth, Butler, & Shah, 2020; Su, 2020; Zahmatkesh & Al-Turjman, 2020; Zhu et al., 2019).

Deep learning has recently reached significant success in various application scenarios, including computer vision, natural language processing, and big data analysis. However, the excellent performance is followed by the ever-increasing computation-consumption and memory demand, thus making it problematic for deployment on resource-constrained devices. The training of neural networks requires a huge amount of power resources, which will be equivalent to a huge CO2 emission (Strubell, Ganesh, & McCallum, 2019). Recent years, there has been a big emphasis on presenting the low-cost architectures, e.g. the SqueezeNet (Iandola et al., 2017), MobileNet (Howard et al., 2019, 2017; Sandler, Howard, Zhu, Zhmoginov, & Chen, 2018), ShuffleNet (Zhang, Zhou, Lin, & Sun, 2018), Xception (Chollet, 2017), GhostNet (Han et al., 2020). But manually designing more efficient architectures relies heavily on human experts' experience, let alone design the network under resource-constrained, e.g., the latency target and memory-consumption limit.

Different from promoting the compact models by designing, some compress-accelerate methods, e.g. network pruning (Han, Pool, Tran, & Dally, 2015; He, Zhang, & Sun, 2017; Li, Kadav, Durdanovic, Samet, & Graf, 2017; Lin, Rao, Lu, & Zhou, 2017; Liu et al., 2017), quantization (Courbariaux, Bengio, & David, 2015; Courbariaux, Hubara, Soudry, El-Yaniv, & Bengio, 2016; Li & Liu, 2016; Rastegari, Ordonez, Redmon, & Farhadi, 2016; Zhou et al., 2016) and the knowledge distillation approach (Hinton, Vinyals, & Dean, 2015) are also widely employed to achieve compact models.

Since the significant success of NAS is first reported by Zoph and Le (2017) and Zoph, Vasudevan, Shlens, and Le (2018), it has opened up a path towards the trade-off between high efficiency and well performance. Especially, the multi-objective NAS (Dong, Cheng, Juan, Wei, & Sun, 2018; Hsu et al., 2018), and Pareto-NASH (Elsken, Metzen,

* Correspondence to: No.2006, Xiyuan Ave, West Hi-Tech Zone, Chendu 611731, China.
E-mail addresses: longfeilu@std.uestc.edu.cn (L. Lu), blyucs@outlook.com (B. Lyu).

https://doi.org/10.1016/j.scs.2021.102747
Received 29 December 2020; Received in revised form 20 January 2021; Accepted 24 January 2021
Available online 29 January 2021

& Hutter, 2018) optimize the multiple objectives that comprehensively involve the precision, parameters, and FLOPs during the search procedure. This literature experimentally achieves the Pareto-optimal solutions and facilitates the feasibility of deploying models.

Although the Parameters and FLOPs are widely exploited and easily obtained evaluation indicators, some literature has reported that the number of Parameters and FLOPs cannot directly reflect the latency, especially considering the difference of running platform. For example, MobileNet (Howard et al., 2017) and NASNet (Zoph et al., 2018) have similar FLOPs (575M vs. 564M), but their latencies are significantly different (113 ms vs. 183 ms) (Tan et al., 2019). To address these issues, MnasNet (Tan et al., 2019) proposed the platform-aware neural architecture search that directly employed the real-world inference latency to the reinforcement learning objective.

It is obvious that online latency measurement is a cost-prohibitive way to be spread out, there are two primary reasons:

- Time cost: We empirically find that averaging dozens of inference runs is necessary to produce an accurate latency measurement, which inevitably brings up numerous time cost. The NAS process is sequentially serialized as a loop of sampling, training, deployment, and inference. That is, the inference latency data is expected to reversely supervise the decision of the search strategy. Thus, most of the time cost of the whole pipeline of NAS will be wasted on this.
- Energy cost: The deep learning server has a power level of kilowatt, and a lot of energy will be wasted while waiting for the delay evaluation. In general, inference latency is very useful for the architecture search, but it also brings up a huge energy expense and an environmental and social burden.

In light of this, the novel ProxylessNAS (Cai, Zhu, & Han, 2019) proposes the layer-wise latency predictor, which builds up a look-up table to estimate the latency for the resource-constrained NAS. But in our view, we believe that the latency of a deep CNN model should not be calculated with simple hierarchical addition, especially for complex and heterogeneous network structures, which are extremely common out of NAS. It assumes sequential processing of operations that may not represent the model architecture and hardware characteristics that affect the end-to-end latency, e.g., whether operations can be computed in parallel on the target hardware. Consequently, we propose an offline end-to-end latency prediction framework to enable NAS. To demonstrate this, we experiment with various machine learning and deep learning regression models on it based on the dataset we establish.

In overall, our contribution can be specified as:

- To reduce the energy consumption of the widely used NAS process in industry and research, we first propose the end-to-end latency prediction framework.
- We carry out the novel encoding methods on MobileNetV3-like search space and sample the latency dataset.
- We comprehensively conduct the regression experiments with several machine learning and deep learning models, including SVR, LR, GBDT and LSTM, and so on. Eventually, our best model achieves fantastic results, with **MSE = 0.9349**, **MAE = 0.5249**, $\boldsymbol{R^2}$ **= 0.9842 and corrcoef = 0.9925**. Besides, our framework is of great scalability and can be applied in all the MobileNetV3-based models.
- Taking several classic multi-objective NAS as the baseline, we analytically find that, for only one deployment by NAS, for example, the NEMO (Kim, Reddy, Yun, & Seo, 2017), it will save **1588 kWh·PUE energy, 1515 pounds CO2 emissions, and $3176 cloud compute cost of AWS**.

2. Related work

Energy consumption and CO2 emission of deep learning. The ever-increasing deep learning models have gained notable progress in accuracy across various artificial intelligence tasks, e.g. computer vision, NLP (Cao, Cao, Guo, Huang, & Wen, 2020; Devlin, Chang, Lee, & Toutanova, 2018; Peters et al., 2018; So, Le, & Liang, 2019), and some other tasks. However, the improvement of precision highly relies on extremely large computational resources, which represents the inevitable tremendous energy consumption and CO2 emission. The energy consumption of the intelligent systems and their environmental problems have been widely concerned by academic circles. Zhou, Fang, Xu, Zhang, and Ji (2019) employ the LSTM to predict the energy consumption of the air conditioning systems. Meanwhile, the empowerment of data-driven machine learning and deep learning approaches have brought new research directions to the electricity, building, and energy industries (Seyedzadeh, Pour Rahimian, Rastogi, & Glesk, 2019). It also lays great emphasis on the data science applied to environmental problems (Liu, Wang, Tang, & Tang, 2019).

Model compression and energy saving. To deploy deep learning applications in low computing power devices to reduce energy consumption, some compact architectures are proposed by manual-designed, e.g. SqueezeNet (Iandola et al., 2017), MobileNet (Howard et al., 2019, 2017; Sandler et al., 2018), GhostNet (Han et al., 2020), ShuffleNet (Zhang et al., 2018), Xception (Chollet, 2017). But manually designing more efficient architectures relies heavily on human experts experience, let alone design the network under resource-constrained, e.g., the latency limit and energy-consumption. Different from proposing the low-cost models by designing, some model compression approaches, e.g. network pruning (Han et al., 2015; He et al., 2017; Li et al., 2017; Lin et al., 2017; Liu et al., 2017), quantization (Courbariaux et al., 2015, 2016; Li & Liu, 2016; Rastegari et al., 2016; Wang, Cao, Guo, Huang & Wen, 2020; Wang et al., 2020; Zhou et al., 2016) knowledge distillation (Hinton et al., 2015), are widely used to achieve compact models.

Neural architecture search. The promising progress of neural architecture search was first reported by Zoph and Le (2017) and Zoph et al. (2018), which is proposed on CIFAR-10 and Penn Treebank dataset that stand for the benchmark task of image classification (CIFAR-10, ImageNet) and NLP tasks, respectively. But on these tasks, a mass of computational resource (20,000 GPU-days for the work of Zoph & Le, 2017 and 2000 of Zoph et al., 2018) is required. So some subsequent literature tries to promote the efficiency of the search procedure, for example, ENAS (Pham, Guan, Zoph, Le, & Dean, 2018) proposed in the continuity of previous work (Zoph & Le, 2017; Zoph et al., 2018). In the early days of NAS research, such huge computational overhead (electricity cost, time cost) was beyond the reach of ordinary research institutes and commercial organizations, which inevitably results in numerous CO2 emissions.

Multi-objective neural architecture search. The above search methods do not comprehensively consider the platform constraints in the search process. Our work is most motivated by the DPP-Net (Dong et al., 2018), MONAS (Hsu et al., 2018), MnasNet (Tan et al., 2019), and Pareto-NASH (Elsken et al., 2018), which optimize the multiple objectives, such as both the number of parameters and precision on CIFAR-10 datasets. The search procedure of MnasNet is time-consuming. MnasNet works in discrete space, in which Reinforcement learning is used to train an RNN controller. MONAS (Hsu et al., 2018) and DPP-Net (Dong et al., 2018) are capable of optimizing multi-objective, precision, and other objectives evaluated by the target platforms. But the deployment of real-time data is not feasible to achieve and the time cost spontaneously increase.

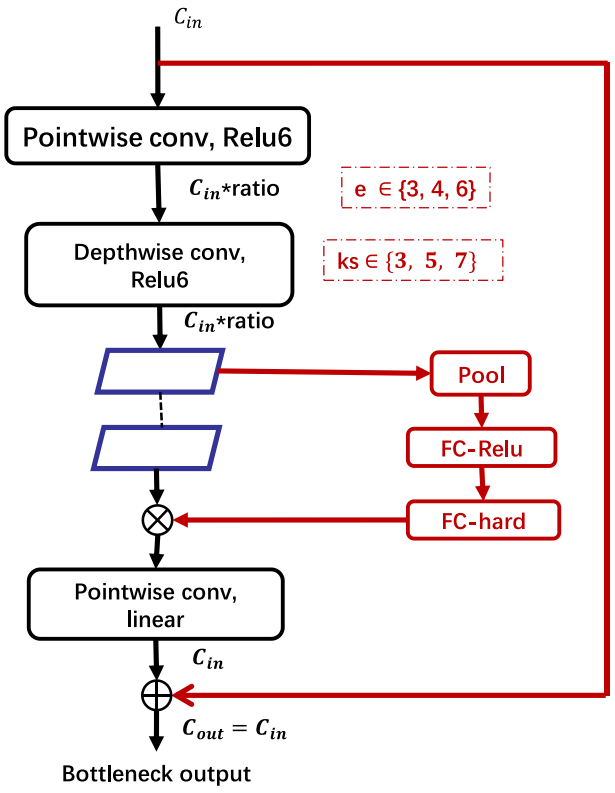

Fig. 1. An overview of MobileNetV3 bottleneck, where ks means kernel size and e means expansion ratio, and both of the two parameters have three choices.

Performance prediction. Some performance prediction methods concentrate on predicting the learning curve of the training accelerating the training (Bowen, Otkrist, Ramesh, & Nikhil, 2016; Domhan, Springenberg, & Hutter, 2015; Klein, Falkner, Springenberg, & Hutter, 2016). But recently, the research has been shifted to the other direction, exploring performance prediction on the basis of properties of the architectures (Liu et al., 2018). Kipf et al. resort to GCN (Kipf & Welling, 2017) to describe the structure of neural architectures as the directed graphs. Similarly, NPENAS (Wei, Niu, Tang, & Liang, 2020) employed GNN-based to accomplish the precision prediction of models. Some benchmark datasets have already been constructed for researchers to study the performance prediction issues (Dong & Yang, 2020; Siems et al., 2020; Ying et al., 2019), which will significantly improve the search efficiency and save much more energy consumption.

Latency prediction. To achieve the easily gained feedback of efficiency, recent works (Cai et al., 2019; Wu et al., 2019) leverage a layer-wise predictor which derives the latency by summing latency measured for each operation in the model individually. In our view, it is not reasonable to assume the latency pattern to be sequential, for it does not take account into the intertwining of the operations and the hardware characteristics that directly affect the end-to-end latency, e.g., operations can be executed in parallel on the numerous devices.

Our work. Motivated by these, we propose the end-to-end latency prediction framework and creatively generate the dataset on target devices with MobileNetV3-like search space. Thus, the latency prediction problem is formulated as the supervising problem, and the coding scheme of the architecture and the learning of the knowledge representation are what we need to explore. We experiment several machine learning/deep learning models, including SVR, LR, BRR, LSTM, and GBDT. And finally, our best model (LGLP) achieves a significant prediction performance. Through our rough analysis, after applying our LGLP model to NEMO, we can save 1515 pounds CO2 emissions and more than 3176 dollars for one deployment, 101 pounds CO2, and 424 dollars for DPP-Net and at most $76896 for MnasNet.

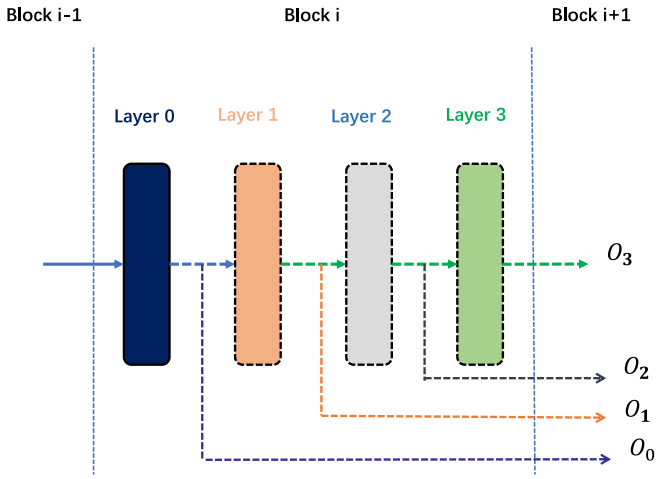

Fig. 2. Depth search bottleneck.

3. Search space and network structure

Our search space is based on MobileNetV3 (MbV3), which combines the operations of depthwise separable convolution, inverted residual with linear bottleneck and Squeeze-and-Excite module. In this way, it can improve efficiency profoundly by reducing computational cost without sacrificing accuracy and increasing latency. Specifically, in our experiments, each model has 10 blocks and each block has the max depth of 4, and the min depth of 1 (As shown in Fig. 2). In other words, the layers of our MbV3 model ranges from 10 to 40. The detailed internal structure of bottleneck is shown in Fig. 1, and there are lots of parameters that we can control, such as kernel size (ks), expansion ratio (e), stride, Squeeze-and-Excite (se), shortcut connection and channels. To be precise,

$$ks \in \{3, 5, 7\}$$

$$e \in \{3, 4, 6\}$$

$$stride \in \{1, 2\}$$

$$se \in \{True, False\}$$

$$shortcut \in \{True, False\}$$

$$channel_{in} \in \{24, 32, 48, 96, 136, 192, 232, 272, 304, 384, 576\}$$

$$channel_{out} \in \{24, 32, 48, 96, 136, 192, 232, 272, 304, 384, 576\}$$

4. Pipeline and dataset generation

4.1. Pipeline

The pipeline of generating dataset is built as Fig. 3 shows. The model construction is accomplished with the input of original architecture parameters. As for deployment on NVIDIA GPU, the model is converted and adapted for TensorRT engine. To reduce the randomness of the inference performance, the sampling process is repeated and the corresponding latencies are averaged.

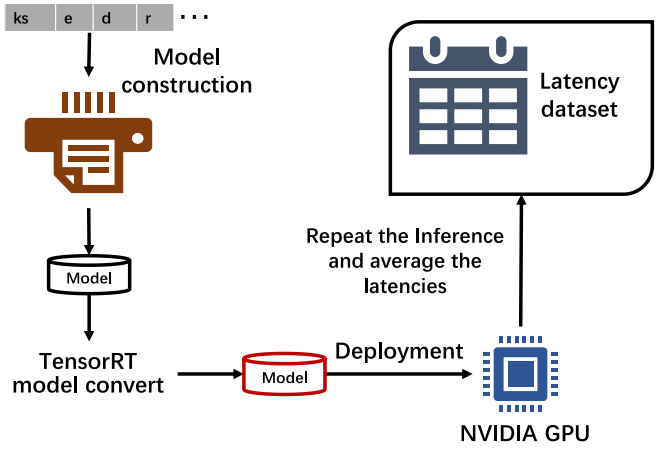

Fig. 3. Pipeline of the generating of the dataset.

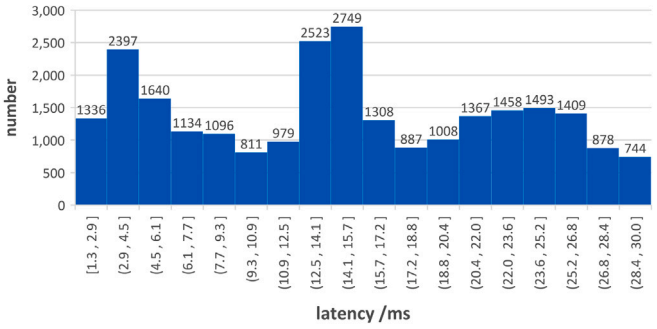

Fig. 4. Final distribution of the dataset.

4.2. Design of the data equilibrium

There is always a saying going like this "garbage in, garbage out", which means that if the meaningless and wrong data is utilized to train our model, you will get the same meaningless and wrong output. Models generated on imbalanced data will not perform well in the real application. It is clear that the quality of the dataset is of great importance when training our model. In the process of collecting the latency dataset, we have taken notes of this problem and taken two measures to avoid and alleviate the disproportion of latency label. Taking into account the tremendous search space, which is because there are lots of parameters in the MbV3 model architecture, and also multiple choices for each parameter. Exactly, the search space contains more than $3^{40} * 3^{40} * 4^{10} * 5 = 10^{45}$, even though we do not have thought of shortcut connection, strides and se. Therefore, it is impractical to traverse the whole search space. Our preliminary experiments show that the magnitude of the latency mainly depends on the resolution of an input image and the depth of the model. Based on the above findings, we take the following two measures.

- We traverse the entire depth and resolution search space and get all the possible model architecture parameters. Then we pick a certain number of samples at random. That is to say, the sample in our dataset has uniform distribution, so does our latency label. After getting the real latency label, we plot the distribution and find that the number of samples is well-proportioned in per unit time when the latency is lower, while the number of samples with larger latency time is relatively few. And we hold the opinion that this is due to we only control the depth and resolution. Which leads to our next action.
- According to the distribution of our dataset established in the first step, we additionally sample those with larger latency time to make up the imbalance. Exactly, we sample those model architectures with larger depth and resolution with a higher probability. Ultimately, we have a preliminary version of the latency dataset. The distribution plot is as Fig. 4 shown.

4.3. Data cleaning

Our experimental observation and data analysis show that although we have obtained reliable delay by averaging latency many times, there are still some architectures that make delay data abnormal due to unknown reasons, such as the running state of graphics card and hardware accelerator state. To illustrate, we picked out some architectures with large delay and tested them again. We found that there was a big discrepancy between the delay and the original data. By extension, there were still a small number of delay data outliers in the whole data. We employ the data cleaning method based on model detection to remove 0.5% of the data.

5. Architecture encoding

In this section, we will describe how the model is encoded in detail, and we will show four kinds of encoding scheme utilized in our LGLP model.

5.1. Encoding scheme A

It is clear that the latency of a model has something with kernel size, expansion ratio, and resolution of the input image. Therefore, the first encoding scheme we have tried is to encode specific parameter figures into a vector. As shown in Fig. 5, if the first layer has the kernel size is 3, expansion is 4, and resolution is 512, then we express it with a 3-dim vector [3, 4, 5.12]. Because the resolution is much greater than kernel size and expansion, we make the resolution divisible by 100. As for those blocks that have layers less than four, we encode the last $(4 - block_{depth})$ layers with zeros. In this case, for each model architecture, we encode it with a 120-dim vector.

5.2. Encoding scheme B

Based on the encoding method used in once-for-all (Cai, Gan, Wang, Zhang, & Han, 2020), we encode our model into a one-hot vector with the dimension of 256, which can be separated into three parts in the whole. In our search space, both the kernel size and expansion have three choices.

$$ks \in \{3,5,7\}, e \in \{3,4,6\}$$

Namely, if the kernel size is 5 in the first layer of the MbV3 model, then we encode it with a three-dimensional vector [0, 1, 0]. However, if the block has a layer of 3, that is to say, the fourth layer does not participate in the process of building the model, and we signify the vector with [0, 0, 0]. The encoding of expansion is just like the kernel size. As for the resolution, in our experiments, it can be chosen from 160, 320, 512, 768, 1024 and we map it into an eight-dimensional one-hot vector by dividing by 146 and choosing the aliquot part.

5.3. Encoding scheme C

Considering the kernel size and expansion in the same layer have specific internal connections, and should not be split into two parts just like the encoding scheme B. Hence, we encode the kernel size and expansion of one layer together with a 6-dim vector rather than encode all the kernel vectors together with all the expansion vectors. Similarly, we populate layers that are not involved in model building with zeros. Besides, we encoded the resolution of the input image into an eight-dim one hot vectors by dividing by 146 and select the aliquot part. In this case, one model can be expressed by 248-dim one-hot vectors.

5.4. Encoding scheme D

The latency time is not only related to kernel size, expansion and resolution but also to the number of channels, stride, squeeze-and-excite connection, and shortcut connection, which are the factors we do not take into account in the first three encoding scheme. In this encoding design, we take an overall consideration between the advantages and disadvantages of B, C encoding schemes. All the architecture hyperparameters of one layer are signified by a 37-dim one-hot vector, and then all the vectors are concatenated into a vector with the dimension of $37 * 40 = 1480$. Identically, we deal with the layers uninvolved in the final model by padding with zero.

6. Model

In this section, we will introduce some background information about some prevailing regression models in our experiments at first. Then, something about long short-term memory networks and how we creatively employ them to deal with our problem will be stated. Lastly, we will present our final LGLP model based on LightGBM and LSTM.

6.1. Machine learning model

In recent years, deep learning has received a lot of attention and favor. And it has made great achievements in various fields, such as natural language processing, image classification (Wang, Cao, Guo, Huang & Wen, 2020), and image segmentation, and so on. In spite of the popularity of the deep learning model, the traditional machine learning algorithms are still very competitive when it comes to their strong interpretability and time-saving characteristic. In our experiments, we also explore several classical and prevailing regression algorithms, such as Support Vector Regression, Linear Regression, Bayesian Ridge Regression, and Elastic-Net Regression. Moreover, we experiment with the popular and powerful gradient boosting decision tree, i.e. Gradient Boosting Decision Tree (GBDT), and its improved version LightGBM, which we will introduce in detail in the following essay.

6.1.1. Support Vector Regression

Support Vector Regression (SVR) is an application of Support Vector Machine on the problem of regression. However, different from Support Vector Machine where we need to find out a hyperplane having the largest gap from the support vector, in SVR we define a parameter ϵ, and all the data in the range of 2ϵ has a residual error of zero. And for those out of the range of 2ϵ, their residual error is the distances to the boundary of the hyperplane. Then, we minimize the sum of all the residual error. In other words, SVR needs to define a hyperplane that makes the farthest point has the smallest distance to the hyperplane. Namely:

$$\min_{b,\boldsymbol{w}}\left(\frac{1}{2}\boldsymbol{w}^{T}\boldsymbol{w}+C\sum_{n=1}^{N}\xi_{n}\right) \tag{1}$$

$$s.t.\left|\boldsymbol{w}^{T}\boldsymbol{z}_{\boldsymbol{n}}+b-y_{n}\right|\leq\epsilon+\xi_{n} \tag{2}$$

$$for\ all\ n,\xi_{n}\geq 0 \tag{3}$$

6.1.2. Linear Regression

Linear Regression is a commonly used statistical analysis method, which resorts to the least square function called the linear regression equation to model the relationship between one or more independent variables and dependent variables. The most essential application for linear regression is to predict the label of new data by means of the model trained from the data that has been observed. And it has been thoroughly studied and get a widespread application in plenty of fields.

6.1.3. Bayesian Ridge Regression

Bayesian linear regression is a linear regression model solved by the Bayesian inference method in statistics. Comparing with the traditional linear regression model, Bayes linear regression regards the parameters of the linear model as random variables, and figures out the posterior probability of model parameters by their prior probability. Bayes Ridge Regression joins the second-order loss of weight coefficient to deal with the problem of overfitting, which is one of the simplest implementations of using Bayesian inference in statistical methods.

6.1.4. Elastic-Net

Elastic-Net is a linear regression model utilizing both the first-order and the second-order loss as regular terms to avoid and alleviate the problem of overfitting. This combination allows for learning a sparse model where few of the weights are non-zero like Lasso, while still maintaining the regularization properties of Ridge, which makes the model capable of learning better when there are multiple features that are correlated with one another.

$$\min_{w}\frac{1}{2n_{samples}}\|Xw-y\|_{2}^{2}+\alpha\rho\|w\|_{1}+\frac{\alpha(1-\rho)}{2}\|w\|_{2}^{2}$$

Similar to Linear Regression, we minimize the total distances of all samples. α is a constant that multiplies the penalty term and ρ is the mixing parameter of Elastic-Net.

6.1.5. Gradient Boosting Decision Tree

Gradient Boosting Decision Tree (Friedman, 2001) realizes the process of learning with combining basic models (Decision Tree) linearly and forward propagation algorithm, i.e. it establishes a new model to fit the residual error of the former model, and then optimizes the new model. After iterating lots of times, GBDT ensembles some of the weak models to gain a powerful and excellent model with diverse strategies. Recently, it is widely used in competitions to got better performance, further. The algorithm is represented in Algorithm 1.

Algorithm 1: Gradient Boosting Decision Tree Algorithm

Initialize $f_0(x) = argmin_\gamma \sum_{i=1}^{N} L(y_i, \gamma)$. **for** $m = 1 \to M$ **do**
  for $i = 1, 2, \to N$ **do**
    $r_{im} = -[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}]_{f=f_{m-1}}$
  end
  Fit a regression tree to the targets r_{im} giving terminal regions $R_{jm}, j = 1, 2, \to J_m$. **for** $j = 1, 2, \to J_m$ **do**

$$\gamma_{jm} = \underset{\gamma}{argmin} \sum_{x_i \in R_{jm}} L(y_i, f_{m-1}(x_i) + \gamma)$$

  end
  Update $f_m(x) = f_{m-1}(x) + \sum_{j=1}^{J_m} \gamma_{jm} I(x \in R_{jm})$.
end
Output $\hat{f} = f_M(x)$.

6.1.6. LightGBM

LightGBM (Light Gradient Boosting Machine) (Ke et al., 2017) is a gradient framework that utilizes tree based on learning algorithms. It is designed by optimizing the problem existing in the traditional Gradient Boosting Decision Tree (GBDT) and Extreme Gradient Boosting (XGBoost) model and performs well with the merits of faster training and higher efficiency, lower memory usage, better accuracy, and support of parallel and GPU learning, which makes it feasible for LightGBM to deal with large-scale data. In our experiments, the encoding vector of the model is fed into the LightGBM model to regress the latency.

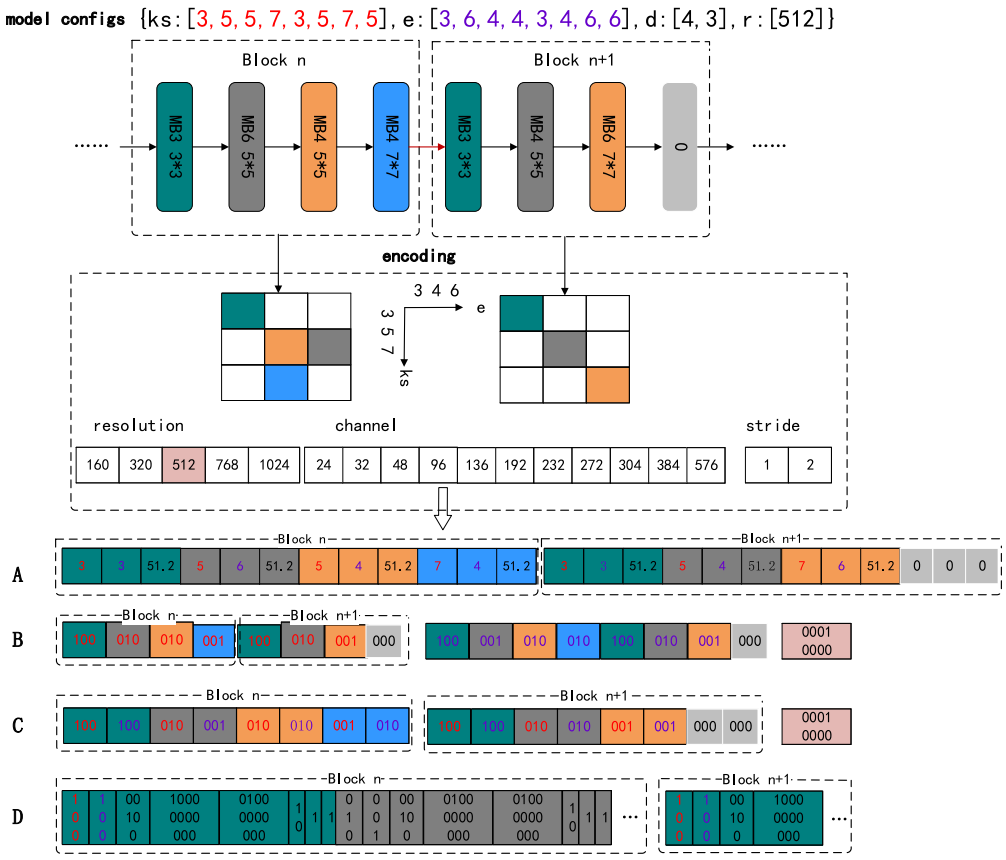

Fig. 5. A detailed explanation of our encoding. We take two blocks for example to illustrate our encoding scheme. Mb6 5*5 means bottleneck with a kernel size of 5*5 and an expansion rate of 6. We use the same color to represent the same bottleneck or parameters. In encoding A, we encode the specific value of parameters directly, while the last three are one-hot encoding. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

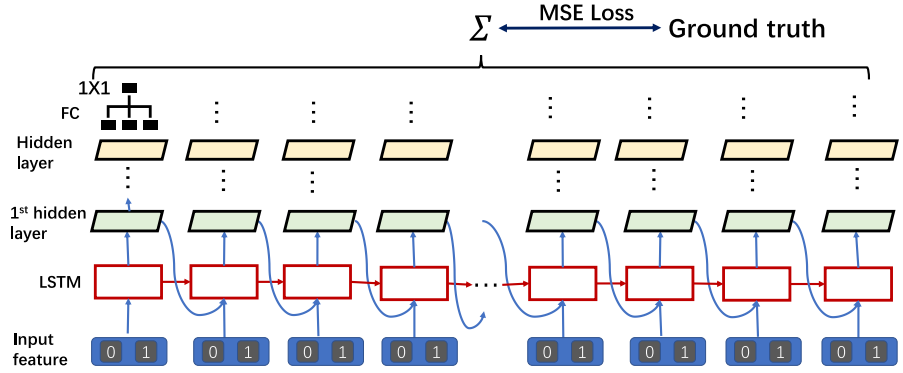

Fig. 6. The proposed regression model based on LSTM.

6.2. Deep learning models

6.2.1. Long Short-Term Memory network

Long Short-Term Memory network, (LSTM) (Hochreiter & Schmidhuber, 1997) is a special kind of Recurrent Neutral Network, and it is made up of plenty of cells having a peculiar internal structure. Comparing with the common recurrent unit, LSTM is capable of encapsulating the notion of forgetting part of its previously-stored memory, as well as adding part of new information, which makes it possible for LSTM to deal with time series based input problem. The MobileNetV3 is a typical representative of the seq2seq model, and the input and output of a specific block are affected by the former block's output. That is to say, predicting the latency time of one MobileNetV3 model from the parameters of all blocks can be viewed as a sequence input problem, which motivates us to have a try on the model LSTM. In our experiment, LSTM is employed to extract the mixed feature, then the output of LSTM is thrown into a regression model. Fig. 6 shows the whole topology structure of the LSTM-based regression model. The core formulas of the propagation of the LSTM are as Eqs. (4)~(9) shown.

$$f_i = \sigma(W_f[h_{i-1}, x_i] + b_f) \tag{4}$$

$$in_i = \sigma(W_{in}[h_{i-1}, x_i] + b_{in}) \tag{5}$$

$$g_i = \tanh(W_g[h_{i-1}, x_i] + b_g) \tag{6}$$

$$out_i = \sigma(W_{out}[h_{i-1}, x_i] + b_{out}) \tag{7}$$

$$c_i = f_i * c_{i-1} + i_i * g_i \tag{8}$$

$$h_i = o_i * \tanh(c_i) \tag{9}$$

where W, b denotes the weights and bias, respectively. in_i, f_i, out_i denote input gate, forget gate, and output gate at time step i, respectively. The g_i and c_i are the cell state and final memory cell state. For each time-step i, h_{i-1} and x_i are the hidden state and the input at corresponding time step.

Specific to our problem of predicting the latency time, it is a regression problem. Innovatively, we propose a new connection mode, which is shown in Fig. 6. Instead of feeding the hidden state of the last cell into the regression layer, we add all the hidden vectors together and feed the sum into the regression layer. The final experiment results show that this connection mode outperforms than only using the last hidden state, or using all the hidden state just by concatenating them ordinarily.

6.3. Our LGLP model

Our LGLP model is based on LSTM and LightGBM, and we innovatively merge the merits of these two models. In our LGLP model, LSTM is regarded as a feature extractor because of its formidable ability to extract features from sequence input, and instead of using the fully connected layer as the classifier, we regress the latency time by means of LightGBM. As far as we are concerned, LightGBM, an machine learning model, is not compatible well with Pytorch because it does not support gradient back-propagation. Therefore, the two basic models are trained respectively. To be precise, we train LSTM with the original dataset we established initially. After LSTM was totally trained, we saved all the model parameters. Then, we got the extracted features (the input of LightGBM) by loading the trained LSTM model and inputting the original dataset once again. In this way, we can change our primary dataset into a new virtual dataset, which incorporates the relationships of all features. Final experiment results show that LGLP outperforms the basic models greatly. Fig. 7 shows the whole topology structure of LGLP model.

7. Experiments

Based on the proposed dataset, we carry out contrast experiments among machine learning models, deep learning models, and our LGLP. In this section, we will introduce our experiment settings about our hardware and relevant parameters firstly. Next, the experiment results will be displayed.

7.1. Experiment settings

We resort to the LSTM as the initial part of our LGLP model, and the LSTM has three layers and binary directions with the hidden size = 64. When training LSTM, we use the popular Adam as our optimizer with learning rate = 0.001 and weight decay = $3e^{-4}$ and the loss function is standard MSE loss. After training for 100 epochs with the batch size = 128, we fetch the last hidden state of LSTM output. Next, we begin to train LightGBM with boosting_type = gbdt, num_leaves = 500, learning rate = 0.01 and n_estimators = 3000. Besides, because of the power learning ability of LightGBM, we continue training until the MSE loss of valid dataset does not improve for 15 rounds to prevent overfitting. The entire experiments are implemented based on Pytorch = 1.5.0, TensorRT = 7.2.1.6 on Ubuntu 16.04, with the devices of Intel(R) Xeon(R) CPU E5-2620 v4 @ 2.10 GHz and four NVIDIA GeForce GTX 1080Ti GPUs.

Table 1
Comparison of the prediction results of various models in different metrics.

Model	MSE	MAE	R^2	corrcoef
Support Vector Regression	4.2377	1.1651	0.9181	0.9657
Linear Regression	5.0040	1.6305	0.9092	0.9573
Bayesian Ridge Regression	4.9665	1.6239	0.9084	0.9576
Elastic-Net	22.5077	4.0769	−0.2444	0.9237
GBDT	4.4234	1.5122	0.9165	0.9629
LightGBM	0.9946	0.5413	0.9830	0.9922
LSTM, layers = 1	2.0209	1.0719	0.9658	0.9830
LSTM, layers = 1, bidirectional	1.8187	0.9712	0.9693	0.9849
LSTM, layers = 3, bidirectional	1.2922	0.7861	0.9782	0.9892
LGLP	**0.9349**	**0.5249**	**0.9842**	**0.9925**

7.2. Evaluation metrics

- Mean absolute error (MAE) is the mean absolute error of all samples, and it can clearly reflect the error between predicted values and true labels. It can be calculated by the following equation, where y_i, f_i are labels and predictions, respectively (The same goes for other equations).

$$MAE = \frac{1}{m}\sum_{i=1}^{m}|f_i - y_i|$$

- Mean squared error (MSE) measures the mean square distance between the observed values and true labels, which is used in regression problem to evaluate model performance. It is figured out by the following equation.

$$MSE = \frac{1}{m}\sum_{i=1}^{m}(f_i - y_i)^2$$

- Coefficient of determination, also known as R^2, reflects the degree to which the independent variable explains the change of the dependent variable. Apparently, the closer R^2 is to 1, the better the model fits.

$$R^2 = 1 - \frac{\sum_{i=1}^{m}(f_i - y_i)^2}{\sum_{i=1}^{m}(y_i - \bar{y})^2}$$

$$where\ \bar{y} = \frac{1}{m}\sum_{i=1}^{m} y_i$$

- Correlation coefficient (corrcoef) is used to measure the correlation of two variables, which ranges from −1 to 1, and the closer corrcoef is to 1 or −1, the stronger the correlation.

$$corrcoef = \frac{\sum_{i=1}^{m}(x_i - \bar{x})(f_i - \bar{y})}{\sqrt{\sum_{i=1}^{m}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{m}(f_i - \bar{y})^2}}$$

$$where\ \bar{x} = \frac{1}{m}\sum_{i=1}^{m} x_i,\ \bar{y} = \frac{1}{m}\sum_{i=1}^{m} y_i$$

7.3. Results

Table 1 reports the results of all the models based on the dataset of GPU latency. We show the mean squared error (MSE), mean absolute error (MAE), coefficient of determination (R^2) and correlation coefficient (corrcoef). Besides experimenting on the machine learning models we discussed above, we also test multiple layers and bidirectional LSTM, which have an important impact on the results. From the table, we can clearly see that our LGLP outperforms other models significantly with the MAE only **0.5249**, which is a **half** of LSTM, **one third** of GBDT and even **one-eighth** of Elastic-Net. And the scatter diagram between labels and predictions is shown in Fig. 8.

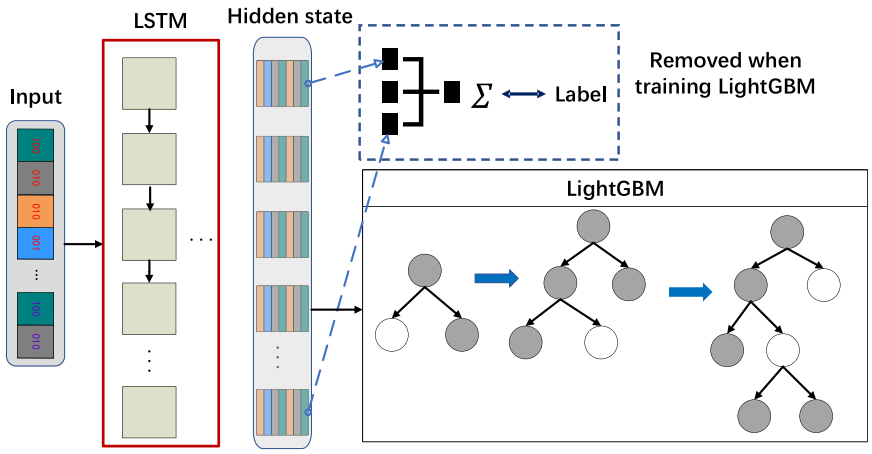

Fig. 7. An overview of LGLP. The part in the dashed frame is the latency predictor we deployed initially, which will be removed when training LightGBM.

Table 2
Estimated CO2 emissions and cloud compute cost, which can be saved by our LGLP. Frequency means the times needed for sampling a model satisfying the target constraint, and "–" means that there is no officially published power data. The cloud compute cost is based on the unit cost of U.S standard, preemptible ($1.46/h–$2.48/h), and on-demand($4.50/h–$8/h) for min and max cost. Notably, the cloud cost of NVIDIA 1080Ti and Tesla M40 is calculated by converting them into TPUv2 required according to their computational capabilities.

Model	Hardware	Power	Frequency	Time saved	kWh·PUE	CO2e	Cloud cost
MnasNet	64 TPUv2	–	8000	267 h	–	–	$23073–$76896
DPP-Net	4 GTX 1080Ti	1 Kw	2000	67 h	105	101	$424
NEMO	60 T M40	15 Kw	2000	67 h	1588	1515	$3176

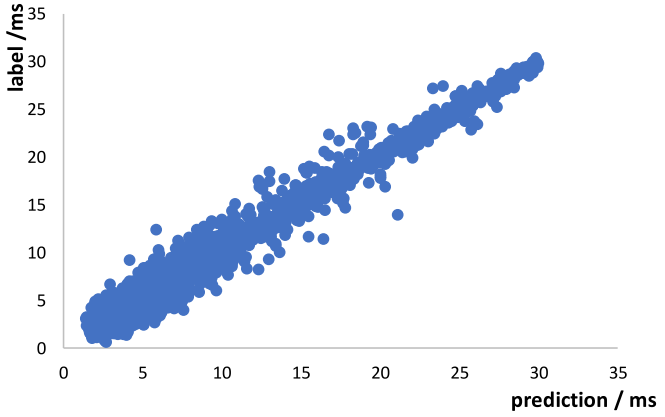

Fig. 8. Prediction result. The horizontal axis is the predicted value, and the vertical axis is the label.

8. Energy saving and benefits

Usually, the neural architecture search samples architectures randomly by means of evolutionary algorithm or reinforcement learning algorithm, and this needs lots of trying. Besides, the host model will have to wait for the detailed latency time to help itself execute the next decision. As is known, the host model always requires high-power computing equipment. When waiting for the feedback from deployed equipment, there will be lots of energy wasted meaninglessly. To quantify the computational energy and environmental cost when training Neural Architecture Search and deploying for getting inference time. Similar to the method in Strubell et al. (2019), we calculate the electricity consumption in training according to the hardware and training time reported in the original paper. Then we multiply the coefficient of CO2 generated (per kilowatt-hour) published by the Environment Protection Agency in 2018 (Emissions, 2018), namely,

$$CO_2e = 0.954p$$

Specifically, we figure out the deployment and inference time, which can be saved by our LGLP model directly about some popular models, such as, DPP-Net, NEMO, and MnasNet. We experimentally found that, e.g., a 30 layers MBV3 model, transferring Pytorch model to onnx then to TensorRT, costs 120 s or so (based on our hardware, which has been stated in experiment setting section). In our estimation, we assume that getting an architecture satisfying latency constraint needs 2000 times, which is an underestimate because the DPP-Net samples 8000 times to get the target model as reported in the primitive paper. The detailed results are listed in Table 2.

9. Conclusion

We proposed LGLP, the first end-to-end latency prediction framework that can precisely and fast predict the latency of one model. Based on the dataset we established and the encoding scheme we designed, our LGLP model can achieve excellent performance, and the MSE, MAE, R^2 and corrcoef between ground-truth and predicted latency on the test set are **0.9349, 0.5249, 0.9842 and 0.9925**, respectively, which is precise enough to abandon deploying on the platform for getting latency. Specifically, according to our rough estimation, after applying our LGLP model to NEMO, we can save 1515 pounds CO2 emissions and more than 3176 dollars for one deployment, 101 pounds CO2, and 424 dollars for DPP-Net and at most $76896 for MnasNet. In this case, our model will save amounts of CO2 emissions and electricity consumption when applying NAS and will bring incalculable benefits to society and the environment.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

Bowen, B., Otkrist, G., Ramesh, R., & Nikhil, N. (2016). *Accelerating neural architecture search using performance prediction*.

Cai, H., Gan, C., Wang, T., Zhang, Z., & Han, S. (2020). Once for all: Train one network and specialize it for efficient deployment. In *International conference on learning representations*.

Cai, H., Zhu, L., & Han, S. (2019). Proxylessnas: Direct neural architecture search on target task and hardware. In *International conference on learning representations*.

Cao, Y., Cao, Y., Guo, Z., Huang, T., & Wen, S. (2020). Global exponential synchronization of delayed memristive neural networks with reaction–diffusion terms. *Neural Networks*, *123*, 70–81.

Chollet, F. (2017). Xception: Deep learning with depthwise separable convolutions. In *CVPR*.

Courbariaux, M., Bengio, Y., & David, J.-P. (2015). Binaryconnect: Training deep neural networks with binary weights during propagations. In *Advances in neural information processing systems 28* (pp. 3123–3131).

Courbariaux, M., Hubara, I., Soudry, D., El-Yaniv, R., & Bengio, Y. (2016). Binarized neural networks: Training deep neural networks with weights and activations constrained to 1 or −1. arXiv: Learning.

Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. N. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: Human language technologies, volume 1 (long and short papers)* (pp. 4171–4186).

Domhan, T., Springenberg, T. J., & Hutter, F. (2015). Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves. In *IJCAI* (pp. 3460–3468).

Dong, J.-D., Cheng, A.-C., Juan, D.-C., Wei, W., & Sun, M. (2018). Dpp-net: Device-aware progressive search for Pareto-optimal neural architecture. In *ECCV*.

Dong, X., & Yang, Y. (2020). *Nas-bench-201: Extending the scope of reproducible neural architecture search*.

Elsken, T., Metzen, H. J., & Hutter, F. (2018). Multi-objective architecture search for cnns. arXiv: Machine Learning.

Emissions, E. (2018). *Generation resource integrated database (egrid)*. Washington, DC: US Environmental Protection Agency.

Friedman, J. H. (2001). Greedy function approximation: A gradient boosting machine. *Annals of Statistics*, 1189–1232.

Han, S., Pool, J., Tran, J., & Dally, J. W. (2015). Learning both weights and connections for efficient neural networks. In *neural information processing systems* (pp. 1135–1143).

Han, K., Wang, Y., Tian, Q., Guo, J., Xu, C., & Xu, C. (2020). Ghostnet: More features from cheap operations. In *2020 IEEE/CVF conference on computer vision and pattern recognition* (pp. 1580–1589).

He, Y., Zhang, X., & Sun, J. (2017). Channel pruning for accelerating very deep neural networks. In *ICCV* (pp. 1398–1406).

Hinton, E. G., Vinyals, O., & Dean, J. (2015). Distilling the knowledge in a neural network. *CoRR*.

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, *9*(8), 1735–1780.

Howard, A., Sandler, M., Chu, G., Chen, L.-C., Chen, B., Tan, M., et al. (2019). Searching for mobilenetv3. In *Proceedings of the IEEE international conference on computer vision* (pp. 1314–1324).

Howard, G. A., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., et al. (2017). Mobilenets: Efficient convolutional neural networks for mobile vision applications. arXiv: Computer Vision and Pattern Recognition.

Hsu, C.-H., Chang, S.-H., Juan, D.-C., Pan, J.-Y., Chen, Y.-T., Wei, W., et al. (2018). Monas: Multi-objective neural architecture search using reinforcement learning. arXiv: Learning.

Iandola, F., Han, S., Moskewicz, M. W., Ashraf, K., Dally, W. J., & Keutzer, K. (2017). Squeezenet: Alexnet-level accuracy with 50x fewer parameters and <0.5 mb model size. arXiv: Computer Vision and Pattern Recognition.

Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., et al. (2017). Lightgbm: A highly efficient gradient boosting decision tree. In *Advances in neural information processing systems* (pp. 3146–3154).

Kim, Y.-H., Reddy, B., Yun, S., & Seo, C. (2017). Nemo: Neuro-evolution with multiobjective optimization of deep neural network for speed and accuracy. In *ICML 2017 AutoML workshop*.

Kipf, N. T., & Welling, M. (2017). Semi-supervised classification with graph convolutional networks. In *International conference on learning representations*.

Klein, A., Falkner, S., Springenberg, J. T., & Hutter, F. (2016). *Learning curve prediction with Bayesian neural networks*.

Li, H., Kadav, A., Durdanovic, I., Samet, H., & Graf, P. H. (2017). Pruning filters for efficient convnets. In *International conference on learning representations*.

Li, F., & Liu, B. (2016). Ternary weight networks. *CoRR*.

Lin, J., Rao, Y., Lu, J., & Zhou, J. (2017). Runtime neural pruning. In *Advances in neural information processing systems 30* (pp. 2178–2188).

Liu, Z., Li, J., Shen, Z., Huang, G., Yan, S., & Zhang, C. (2017). Learning efficient convolutional networks through network slimming. In *ICCV* (pp. 2755–2763).

Liu, Z., Wang, F., Tang, Z., & Tang, J. (2019). Predictions and driving factors of production-based CO_2 emissions in Beijing, China. *Sustainable Cities and Society*, *53*, Article 101909.

Liu, C., Zoph, B., Neumann, M., Shlens, J., Hua, W., Li, L.-J., et al. (2018). Progressive neural architecture search. In *Proceedings of the European conference on computer vision* (pp. 19–34).

Owyer, E., Pan, I., Charlesworth, R., Butler, S., & Shah, N. (2020). Integration of an energy management tool and digital twin for coordination and control of multi-vector smart energy systems. *Sustainable Cities and Society*, *62*, Article 102412.

Peters, M. E., Neumann, M., Iyyer, M., Gardner, M., Clark, C., Lee, K., et al. (2018). Deep contextualized word representations. In *Proceedings of the 2018 conference of the North American chapter of the association for computational linguistics: Human language technologies, volume 1 (long papers)*: Vol. 1, (pp. 2227–2237).

Pham, H., Guan, Y. M., Zoph, B., Le, V. Q., & Dean, J. (2018). Efficient neural architecture search via parameter sharing. In *ICML* (pp. 4092–4101).

Rastegari, M., Ordonez, V., Redmon, J., & Farhadi, A. (2016). Xnor-net: Imagenet classification using binary convolutional neural networks. In *ECCV*.

Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., & Chen, L.-C. (2018). Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 4510–4520).

Seyedzadeh, S., Pour Rahimian, F., Rastogi, P., & Glesk, I. (2019). Tuning machine learning models for prediction of building energy loads. *Sustainable Cities and Society*, *47*.

Siems, J., Zimmer, L., Zela, A., Lukasik, J., Keuper, M., & Hutter, F. (2020). Nas-bench-301 and the case for surrogate benchmarks for neural architecture search. arXiv preprint arXiv:2008.09777.

So, D. R., Le, Q. V., & Liang, C. (2019). The evolved transformer. In *International conference on machine learning* (pp. 5877–5886).

Strubell, E., Ganesh, A., & McCallum, A. (2019). Energy and policy considerations for deep learning in NLP. In *Proceedings of the 57th annual meeting of the association for computational linguistics* (pp. 3645–3650).

Su, Y. (2020). Smart energy for smart built environment: A review for combined objectives of affordable sustainable green. *Sustainable Cities and Society*, *53*, Article 101954.

Tan, M., Chen, B., Pang, R., Vasudevan, V. K., Sandler, M., Howard, A., et al. (2019). Mnasnet:platform-aware neural architecture search for mobile. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 2820–2828).

Wang, Y., Cao, Y., Guo, Z., Huang, T., & Wen, S. (2020). Event-based sliding-mode synchronization of delayed memristive neural networks via continuous/periodic sampling algorithm. *Applied Mathematics and Computation*, *383*, Article 125379.

Wang, S., Cao, Y., Guo, Z., Yan, Z., Wen, S., & Huang, T. (2020). Periodic event-triggered synchronization of multiple memristive neural networks with switching topologies and parameter mismatch. *IEEE Transactions on Cybernetics*.

Wei, C., Niu, C., Tang, Y., & Liang, J. (2020). *Npenas: Neural predictor guided evolution for neural architecture search*.

Wu, B., Dai, X., Zhang, P., Wang, Y., Sun, F., Wu, Y., et al. (2019). Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 10734–10742).

Ying, C., Klein, A., Christiansen, E., Real, E., Murphy, K., & Hutter, F. (2019). Nas-bench-101: Towards reproducible neural architecture search. In *International conference on machine learning* (pp. 7105–7114).

Zahmatkesh, H., & Al-Turjman, F. (2020). Fog computing for sustainable smart cities in the IoT era: Caching techniques and enabling technologies - an overview. *Sustainable Cities and Society*, *59*, Article 102139.

Zhang, X., Zhou, X., Lin, M., & Sun, J. (2018). Shufflenet: An extremely efficient convolutional neural network for mobile devices. In *Computer vision and pattern recognition*.

Zhou, C., Fang, Z., Xu, X., Zhang, X., & Ji, Y. (2019). Using long short-term memory networks to predict energy consumption of air-conditioning systems. *Sustainable Cities and Society*, *55*, Article 102000.

Zhou, S., Wu, Y., Ni, Z., Zhou, X., Wen, H., & Zou, Y. (2016). Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients. arXiv: Neural and Evolutionary Computing.

Zhu, J., Shen, Y., Song, Z., Zhou, D., Zhang, Z., & Kusiak, A. (2019). Data-driven building load profiling and energy management. *Sustainable Cities and Society*, *49*, Article 101587.

Zoph, B., & Le, V. Q. (2017). Neural architecture search with reinforcement learning. In *International conference on learning representations*.

Zoph, B., Vasudevan, V., Shlens, J., & Le, V. Q. (2018). Learning transferable architectures for scalable image recognition. In *Computer vision and pattern recognition*.