machine learning & knowledge extraction

MDPI

Article

Low Cost Evolutionary Neural Architecture Search (LENAS) Applied to Traffic Forecasting †

Daniel Klosa * and Christof Büskens

WG Optimization and Optimal Control, Center for Industrial Mathematics, University of Bremen, 28359 Bremen, Germany; bueskens@uni-bremen.de

* Correspondence: dklosa@uni-bremen.de

† This paper is an extended version of our paper published in the Proceedings of the 21st IEEE International Conference on Machine Learning and Applications (ICMLA), Nassau The Bahamas, 12–14 December 2022.

Abstract: Traffic forecasting is an important task for transportation engineering as it helps authorities to plan and control traffic flow, detect congestion, and reduce environmental impact. Deep learning techniques have gained traction in handling such complex datasets, but require expertise in neural architecture engineering, often beyond the scope of traffic management decision-makers. Our study aims to address this challenge by using neural architecture search (NAS) methods. These methods, which simplify neural architecture engineering by discovering task-specific neural architectures, are only recently applied to traffic prediction. We specifically focus on the performance estimation of neural architectures, a computationally demanding sub-problem of NAS, that often hinders the real-world application of these methods. Extending prior work on evolutionary NAS (ENAS), our work evaluates the utility of zero-cost (ZC) proxies, recently emerged cost-effective evaluators of network architectures. These proxies operate without necessitating training, thereby circumventing the computational bottleneck, albeit at a slight cost to accuracy. Our findings indicate that, when integrated into the ENAS framework, ZC proxies can accelerate the search process by two orders of magnitude at a small cost of accuracy. These results establish the viability of ZC proxies as a practical solution to accelerate NAS methods while maintaining model accuracy. Our research contributes to the domain by showcasing how ZC proxies can enhance the accessibility and usability of NAS methods for traffic forecasting, despite potential limitations in neural architecture engineering expertise. This novel approach significantly aids in the efficient application of deep learning techniques in real-world traffic management scenarios.

Keywords: neural architecture search; traffic forecasting; zero-cost proxies

Citation: Klosa, D.; Büskens, C. Low Cost Evolutionary Neural Architecture Search (LENAS) Applied to Traffic Forecasting. *Mach. Learn. Knowl. Extr.* **2023**, *5*, 830–846. https://doi.org/10.3390/make5030044

Academic Editors: Moamar Sayed-Mouchaweh, Mohd Arif Wani, Vasile Palade and Mehmed M. Kantardzic

Received: 13 June 2023
Revised: 17 July 2023
Accepted: 24 July 2023
Published: 28 July 2023

1. Introduction

Forecasting future traffic conditions, such as flow and speed, by analyzing historical traffic patterns is an essential task for transportation engineering. Accurate traffic forecasts can detect congestion and help authorities plan and control traffic flow, enabling intelligent traffic systems (ITS) to adjust to future events, leading to more uniform traffic flow, and reduced CO_2 emissions, ultimately reducing environmental impact. With the increasing availability of traffic data, there has been a growing interest in developing machine learning algorithms for traffic forecasting.

However, traffic forecasting poses several challenges. Future traffic conditions at a single measurement site depend not only on recent conditions in the temporal dimension but also on upstream measurements in the spatial dimension. Additionally, large datasets with hundreds of measurement sites and complex road networks can make modeling dependencies between them difficult.

Linear regression [1], auto-regressive moving average [2], vector auto-regression [3], and k-nearest neighbors [4,5] are traditional methods that fall short in capturing the complex

spatio-temporal dependencies present in large traffic datasets. As a result, recent research has shifted towards deep learning models such as long-short term memory and gated recurrent unit [6] to capture temporal dependencies, and graph convolutional neural networks (GCN) to learn spatial dependencies within the data [7–9].

One limitation of GCN models is their reliance on prior knowledge of the spatial connections within the graph-structured road network, typically in the form of an adjacency matrix. GraphWaveNet [7] and AGCRN [10] address this issue by learning spatial dependencies directly from the data. However, these methods still rely on handcrafted neural architectures designed by experts. Moreover, deploying these approaches in real-world applications requires additional customization for the specific scenario, which can be time-consuming and require considerable effort.

Neural architecture search (NAS) methods have gained popularity in recent years for discovering customized neural architectures for various tasks, reducing the tedious process of neural architecture design. While early NAS frameworks focused on computer vision and language modeling [11,12], they can also be applied to graph data [13,14] and spatio-temporal data [15].

NAS typically involves three components: the search space, search strategies, and performance estimation. The search space defines the general structure of discovered network architectures, including the operations and their connections within the network. Different search strategies exist, such as reinforcement learning (RL), gradient-based search, and evolutionary NAS (ENAS). RL-based algorithms are known to require significant computational resources, even on smaller datasets such as CIFAR-10 [16]. Gradient-based NAS methods, such as those used in [12], are more efficient but can become trapped in local minima and require the construction of a supernet in advance. ENAS can explore the search space more thoroughly without a given supernet, but can also suffer from long computation times. The performance estimation strategy defines how discovered architectures are evaluated. Typically, this involves training them for a certain number of epochs, which can be costly. However, techniques such as weight inheritance and network morphisms eliminate the need for training from scratch, greatly reducing training time [17]. It is also possible to evaluate discovered architectures without training at all [18,19].

The research on NAS for traffic forecasting is limited. Early work [20] investigates the implementation of genetic algorithm (GA) for optimizing gradient descent hyperparameters and hidden layers in multi layer perceptrons (MLP) on a small dataset. Rahimipour et al. [21] search for number of neurons in two layer MLPs and slopes of the activation functions using GA on a small (three measurement sites) real-world dataset. A particle swarm optimization algorithm is used in [22] to optimize the amount of neurons in the hidden layers of deep belief networks, learning rate and momentum. However, they also limit their study to a small dataset and fix the amount of hidden layers. To our knowledge, Pan et al. [15] are the first to implement gradient-based NAS for traffic forecasting in their framework called AutoSTG. They are using a cell-based approach of learning one smaller architecture (a cell) and applying it in sequence multiple times to obtain a larger network, similar to [12]. Their operation space is made up of none, identity, temporal convolution, and spatial graph convolution. Additionally, they apply meta learning to learn the adjacency matrices for the spatial graph convolutions and kernels for their temporal convolutions. To our knowledge, Klosa and Büskens [23] are the first to apply ENAS for the task of traffic forecasting on four real world datasets. They use a simple genetic algorithm to search through an architecture space flexible in size. Their algorithm does not make use of performance estimation strategies, which leads to tremendous computation times, rendering their approach unfit for application in the real world.

Our study proposes to advance the field by integrating zero-cost (ZC) proxies into the architecture search algorithm used by Klosa and Büskens [23]. ZC proxies offer the advantage of being able to rank neural architectures within the search space without necessitating expensive training [18,19]. This technique has shown promising results in image classification, natural language processing, and computer vision. Our proposed

application of ZC proxies to spatio-temporal data forecasting and specifically to traffic data is therefore novel.

However, it is crucial to acknowledge the potential limitations of this approach. ZC proxies have predominantly been tested in fields other than regression tasks, and their effectiveness in traffic forecasting is not yet fully understood. Additionally, some potential challenges may arise in terms of biases towards architecture size, stability with regard to weight initialization and mini-batch sampling, and the correlation between ZC proxies and validation loss.

To rigorously investigate these issues, our research will aim to answer the following questions:

1. Are ZC proxies biased towards architecture size?
2. Are ZC proxies stable with regards to weight initialization and mini-batch sampling?
3. Are ZC proxies stable with regards to architecture size?
4. Are ZC proxies and validation loss correlated?

Addressing these questions will provide a deeper understanding of the capabilities and limitations of ZC proxies in the context of traffic forecasting, ultimately helping to determine whether this method can be applied reliably in real-world settings.

This research paper is structured as follows; In Section 2 we define the problem of traffic forecasting, introduce the bilevel optimization problem NAS solves, the architecture search space, the genetic algorithm and define the ZC proxies examined in this work. We answer the above mentioned research questions in Section 2.6. We describe the experimental setup of our low cost evolutionary neural architecture search (LENAS) in Section 2.7 and evaluate the performance on four real world datasets in Section 3. In Section 4, we discuss the results, outline possible directions for future work before comming to a conclusion.

2. Materials and Methods

In this section, we describe the task of traffic forecasting. Afterwards, we define neural architecture search and the components making up our framework LENAS. For that, we define the search space, the search method and ZC proxies as our performance estimation. Afterwards we answer the above stated research questions and describe the experimental setup of the LENAS framework.

2.1. Traffic Forecasting

Let $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathbf{W})$ denote an undirected graph, where $\mathcal{V}$ is a set of vertices or nodes representing the $\mid \mathcal{V} \mid = N$ measurement sites in the road network, $\mathcal{E}$ a set of edges indicating the connectivity between measurement sites and $\mathbf{W} \in \mathbb{R}^{N \times N}$ a weighted adjacency matrix, representing the proximity between nodes. Then, given a timestamp t, the traffic conditions on the graph $\mathcal{G}$ are denoted by a graph signal $X_t \in \mathbb{R}^{N \times F}$, where $F \in \mathbb{N}$ is the amount of features observed at each measurement site or node. Finally, the goal of traffic forecasting is to learn a function f for predicting future T graph signals on the graph $\mathcal{G}$ from T' historical graph signals:

$$\mathbf{y}_t = [X_{t+1}, ..., X_{t+T}] = f_\theta([X_{t-T'+1}, ..., X_t]; \mathcal{G}) \in \mathbb{R}^{N \times D \times T}$$

Here, $D \in \mathbb{N}$ denotes the amount of features to predict for each measurement site.

2.2. Neural Architecture Search

The objective of NAS is to find an optimal architecture A from the space of architectures $\mathcal{A}$ that minimizes the loss function $\mathcal{L}$ on a given dataset $\mathcal{D}$. To be more precise, we want to solve a bilevel optimization problem:

$$A^* = \min_{A \in \mathcal{A}} \mathcal{L}(\theta^*(A), A, \mathcal{D}_{\text{valid}}) \tag{1}$$

$$\text{s.t.} \quad \theta^*(A) = \arg\min_{\theta} \mathcal{L}(\theta, A, \mathcal{D}_{\text{train}}) \tag{2}$$

Here, $\mathcal{D}_{\text{train}} \subset \mathcal{D}$ and $\mathcal{D}_{\text{valid}} \subset \mathcal{D}$ respectively denote the training and validation datasets and θ the network parameters.

To solve the bilevel optimization problem, NAS can be split into three components: the architecture search space, the search method, and the performance estimation, which are described in the following sections.

2.3. Architecture Search Space

Figure 1 shows the architecture search space of our method. As can be seen, we are not using a cell-based search approach as in [15], but evolve whole architectures with varying number $N \in \mathbb{N}$ of nodes. The nodes are ordered in a sequence, forming a directed acyclic graph. Each edge (i, j), $i, j \in \mathbb{N}$ is associated with an operation $o^{(i,j)}$ from the operation space $\mathcal{O}$ mapping the node $x^{(i)}$ to node $x^{(j)}$. To obtain node $x^{(j)}$ all of its preceding nodes are summed up:

$$x^{(j)} = \sum_{i<j} o^{(i,j)}(x^{(i)}), \; j = 2, ..., N.$$

The node $x^{(1)}$ is the input node and the node $x^{(N)}$ is the output node of the network. We apply a 2D 1×1 convolution to a given input $X_t \in \mathcal{D}$ to obtain node $x^{(1)}$ and another 2D 1×1 convolution to the output node $x^{(N)}$ followed by a fully connected layer (FC) to obtain the final output. The number of input and output channels $n_c^{(i)}$ and $n_c^{(j)}$ for each operation $o^{(i,j)}$ is fixed to $n_c^{(i)} = \min(2^{i+1}, 128)$.

The operation space is inspired by existing approaches [8,15], which mainly use convolutional operations. In this work, we use the following operations:

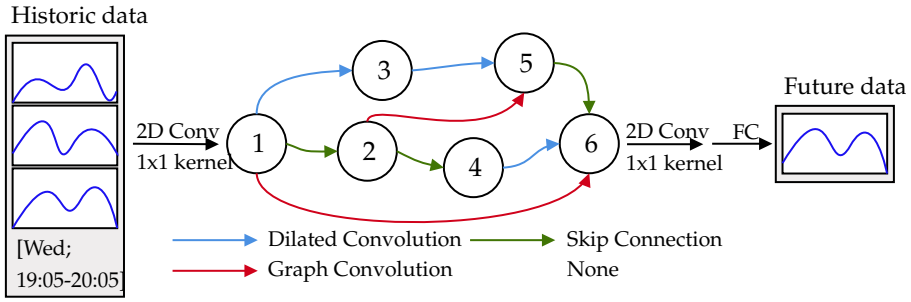

Figure 1. Architecture search space of our framework.

2.3.1. None

The none or zero operation zeroes out the input. This is equivalent to not having a connection between nodes.

2.3.2. Skip Connection

Skip connection usually copies the input. However, since channels differ between nodes, the skip connection employed is a 2D 1×1 convolution that upscales the channel dimension when necessary. This operation has no mutable parameters.

2.3.3. Dilated Causal Convolution

Dilated convolutions are a modification of the standard convolutional operations, designed to increase the receptive field of a network without substantially increasing the number of parameters or the computational cost [24]. Specifically, dilated causal convolutions represent a type of convolution that only allows access to past (causal) information, a feature that is particularly useful when processing sequential or temporal data such as in the cases of time-series prediction or speech synthesis.

In the standard convolutional operations, the elements of the filter are applied to the input elements in a compact, contiguous manner. On the contrary, in dilated convolutions,

the filter is applied to the input with gaps, which are determined by a dilation rate. This leads to an exponential expansion of the receptive field as the size of the filter grows linearly. This combination of causality with the increased receptive field enables the network to efficiently capture long-range temporal dependencies.

Note that the receptive field only increases exponentially when dilation factors increase by a factor of two with each following layer [24]. To fulfill this, we modify the dilation factors manually after each crossover and mutation operation. Mutable parameters are the dilation factor and kernel size.

2.3.4. Graph Convolution

Graph convolutional networks (GCNs) have displayed significant effectiveness in various applications owing to their ability to capture topological correlations present in graph-structured data [7,8,10,15]. Among several methodologies to perform graph convolutions, the method of Chebyshev polynomial approximation has been particularly notable.

The graph convolution operation based on Chebyshev polynomial allows the network to take into account different scales of neighborhood when processing a node in the graph. From the perspective of spectral graph theory, the Chebyshev polynomial approximation has been used to generalize the convolution operation in the Fourier domain, leading to computational efficiency and ensuring a flexible receptive field.

The central concept is to approximate the spectral decomposition of the graph Laplacian, a crucial element of graph Fourier transform, with Chebyshev polynomials. Given a signal x on a graph and a filter defined as a function g_θ of the Laplacian L, the convolution of x with g_θ on the graph is represented in the spectral domain:

$$g_\theta * x = g_\theta(L)x$$

To circumvent the computational overhead associated with the spectral decomposition of the Laplacian, especially for large graphs, the filter g_θ can be approximated using Chebyshev polynomials T_k:

$$g_\theta \approx \sum_{k=0}^{K} \theta_k T_k(\tilde{L})$$

where $\tilde{L} = \frac{2L}{\lambda_{\max}} - I$ is the scaled Laplacian, $\lambda_{\max}$ is the largest eigenvalue of L, and I is the identity matrix. T_k can be computed recursively as:

$$T_k(x) = 2xT_{k-1}(x) - T_{k-2}(x)$$

with $T_0(x) = 1$ and $T_1(x) = x$. Consequently, the filter becomes a K-localized operator, meaning it relies only on the K-hop neighborhood of each node, where K is the order of the polynomial. This method leads to a significant reduction in computational complexity while providing control over the balance between the model's capacity and computational efficiency. K is the mutable parameter in this operation.

2.4. Search Method

We use the same genetic algorithm (GA) as a search method for NAS as in our previous work [11] with the addition of using ZC proxies as performance estimators. The GA is summarised in Algorithm 1. We warmstart the genetic algorithm by choosing a large starting population size n_w and selecting the best n_p performing architectures for the following n_c cycles. For performance estimation of each architecture we use the naswot ZC proxy as described in Section 2.5 due to the performance in the experimental Section 2.6.

Algorithm 1 Genetic algorithm with naswot

Require: $n_w > 0, n_p > 0, n_c > 0$
$population \leftarrow \varnothing$
$best \leftarrow \varnothing$
while $|population| < n_w$ **do** ▷ Warmstart
  $model.arch \leftarrow$ RandomInit()
  $model.fitness \leftarrow$ naswot($model.architecture$)
  add *model* to *population*
end while
$population \leftarrow$ Elitism($population$) ▷ best genomes
for i in $range(n_c)$ **do**
  $offspring \leftarrow \varnothing$
  while $| offspring | < n_p$ **do**
    $parents \leftarrow$ BinaryTournament($population, 2$)
    $children \leftarrow$ UniformCrossover($parents$)
    add *children* to $offspring$
  end while
  for *model* in $offspring$ **do**
    $model.arch \leftarrow$ Mutate($model.architecture$)
    $model.fitness \leftarrow$ naswot($model.architecture$)
    if $model.fitness < best.fitness$ **then**
      $best \leftarrow model$
    end if
  end for
  add $offspring$ to *population*
  $population \leftarrow$ BinaryTournament($population, n_p$)
end for
return *best*

Once the population is initialized and evaluated, the crossover and mutation cycle is repeated $n_g \in \mathbb{N}$ times. We do not use a stopping criterion, but have a fixed amount of cycles. We use binary tournament for selecting two parents for crossover. In binary tournament, two chromosomes are picked at random and the one with better fitness, in our case the ZC proxy score, is selected. Hence, better performing architectures are more likely to be selected, but we still retain diversity. After two parents are selected, we apply uniform crossover by selecting a random subset of nodes in the two parent's architectures to be switched. If the sizes of the architectures are different, we switch a maximum of nodes equal to the amount of nodes in the smaller architecture and retain the sizes. Then, the two resulting children are mutated and added to the current generation. Mutation operations include:

- Switching edges ($o^{(i,j)} \leftrightarrow o^{(i',j')}$)
- Removing an operation ($o^{(i,j)} \leftarrow$ None)
- Changing the type of operation ($o^{(i,j)} \leftarrow o'^{(i,j)}$, where $o' \in \mathcal{O}$ is selected at random)
- Mutating the parameters of an operation (kernel size and/or dilation factor can be increased or decreased to next possible size)
- Adding or removing a node (adding also adds an operation from the operation space, except None)

After mutation, the channels and dilation factors of some operations need to be modified as previously mentioned. All children are then trained and evaluated. Lastly, we use a binary tournament to select n_p models from the current generation to stay in the population.

After n_c cycles, we return the model with the best fitness (naswot score) over all generations.

2.5. Zero-Cost Proxies

As mentioned earlier, performance estimation is the bottleneck of NAS when it comes to computation time. Zero-cost (ZC) proxies aim at evaluating an architectures performance without the need of training, i.e., they evaluate network performance from one forward pass and/or backwards pass of a single mini-batch of random data. In the following, we give a brief overview of the zero-cost proxies used in this work. The ZC proxies snip, grasp, synflow, and Fisher are inspired by pruning theory in which they are used to prune network parameters least contributing to network performance. In recent works they have been applied to score whole neural networks without training [19,25]. The ZC proxies jacob_cov and naswot have been solely designed with scoring networks for NAS in mind. We note that all ZC proxies have been thoroughly investigated for classification tasks [19,25], however, research on regression is to the authors' knowledge non-existent.

2.5.1. Gradient Norm

In gradient norm (grad_norm) the Euclidean norm of the gradients resulting from one mini-batch of data is summed up. Ref. [25] use it in their work on ZC proxies as a baseline.

2.5.2. Single-Shot Network Pruning

Single-shot network pruning (snip) was proposed in [26] for parameter pruning at initialisation stage of neural networks. It was used in [25] as a ZC proxy by computing

$$snip(\theta) = |\frac{\partial \mathcal{L}}{\partial \theta} \odot \theta|$$

for each parameter θ in the architecture A and obtaining the sum $snip(A) = \sum_{\theta \in A} snip(\theta)$. $\odot$ denotes the Hadamard product.

2.5.3. Gradient Signal Preservation

Gradient signal preservation (grasp) was introduced to improve upon snip in [27]. The idea being to incorporate the change of gradient norm instead of loss when pruning a parameter:

$$grasp(\theta) = -\left(H\frac{\partial \mathcal{L}}{\partial \theta}\right) \odot \theta$$

Here, H denotes the Hessian. It was used in [25] as a ZC proxy by computing the sum $grasp(A) = \sum_{\theta \in A} grasp(\theta)$.

2.5.4. Synaptic Flow Pruning

Synaptic flow pruning (synflow) has been introduced as a method of pruning network parameters without the need of training or data [28]. It does so by taking the product of all network parameters as a loss $\mathcal{R}$, avoiding layer collapse:

$$synflow(\theta) = \frac{\partial \mathcal{R}}{\partial \theta} \odot \theta$$

It was used in [25] as a ZC proxy by computing the sum $synflow(A) = \sum_{\theta \in A} synflow(\theta)$.

2.5.5. Fisher

Fisher was introduced in [29] as a method to prune activation channels having the least effect on the loss in a neural network. It computes the sum over all gradients of the activation layers a in an architecture:

$$fisher(a) = \left(\frac{\partial \mathcal{L}}{\partial a} a\right)^2$$

To use it as a ZC proxy, we compute the sum $fisher(A) = \sum_{a \in A} fisher(a)$ as in [25].

2.5.6. Jacob Covariance

Jacobian covariance (jacob_cov) as introduced in [30] measures the flexibility of a neural network by computing the covariance of the Jacobians of the rectified linear units for a minibatch. The idea being that for a network to be able to tell inputs apart the covariance should be low. For more details we refer to the original work [30].

2.5.7. NAS without Training

NAS without training (naswot) is what we will call the successor of jacob_cov as described in [30]. Naswot builds on the same idea, but instead of computing the covariance of the Jacobians, it computes a distance metric based on the activations of the rectified linear units within a network. Given a minibatch $X = \{x_i\}_{i=1}^{b}$ of size $b \in \mathbb{N}$. After a forward pass of the minibatch we obtain the activations of each rectified linear unit a_i. Then, the activations are flattened and converted into a binary code c_i, s.t. $c_{i,m} = 0$ if $a_{i,m} = 0$ and $c_{i,m} = 1$ if $a_{i,m} > 0$. Afterwards, we compute the Hamming distance $d_H(c_i, c_j) \in [0, 1]$ of the binary codes to measure their similarity. We then obtain the matrix

$$K_H = \begin{pmatrix} d_H(c_1, c_1) & \cdots & d_H(c_1, c_b) \\ \vdots & \ddots & \vdots \\ d_H(c_b, c_1) & \cdots & d_H(c_b, c_b) \end{pmatrix}$$

and finally compute the naswot score:

$$naswot(A) = \log|K_H|$$

We use the implementation of [25] for all ZC proxies except naswot, where we used the implementation as in [30] and made some adjustments to make it viable to use for regression tasks instead of classification as intended by the authors.

2.6. Robustness, Bias, and Usability of ZC Proxies

In this section, we aim to answer the research questions with regards to robustness, bias, and usability of zero-cost proxies for NAS in our setting. All of the following experiments are carried out on the PeMSD8 dataset described in Section 2.7.

Since we want to use ZC proxies in genetic algorithm as a measure of fitness, the resulting scores and true fitness should lead to the same or similar ranking within the population. Hence, in the following experiments, we will sample a population with different architectures from our search space and compute the Spearman rank correlation of ZC proxy score and true fitness. The true fitness is determined by training the architectures until they converge and taking the best validation loss. As a loss function, we use the MAE for training.

The main objective is to find a ZC proxy with high correlation to the validation loss. However, there are also questions to be answered when it comes to robustness. We want to find a ZC proxy that is not affected by the weight initialization of the network, nor the sampling of the mini batch used for computation. Furthermore, the choice of channel depths and the size of the mini batch should not affect the score. Previous work [19] has discussed that the size of the architecture, i.e., the amount of layers in a network might have an effect on ZC proxy scoring. To examine this behaviour, apart from the ZC proxy score z, we will also compute

$$z_l = z/n_l, \qquad z_c = z/n_c$$

as the scaled variants of the score by the number of layers n_l and number of channels n_c in each network. The results will answer research question 1 as stated in the Introduction.

2.6.1. Are ZC Proxies Robust with Regards to Weight Initialization and Mini-Batch Sampling?

To answer the second research question, we compute ZC proxy scores for each of 30 sampled architectures on 24 different random seeds. We obtain a ranking of architectures by score for each random seed. Then, we compare the rankings by computing the Spearman rank correlation. Correlation close to 0 indicates that rankings are not correlated, rendering the ZC proxy unusable. Correlation close to 1 (or -1 when negative correlation) indicates similar or same rankings. Additionally, we repeat this process for three different sizes of mini batches (24, 32 and 64). We show the results in Table 1, where we take the mean Spearman rank correlation over the three mini batch sizes.

It can be seen that grad_norm, snip, and synflow obtain good correlations, while naswot performs the best. Scaling by number of layers and number of channels greatly improves the jacob_cov score and improves naswot to reach a perfect correlation. Scaling synflow improves the proxy slightly and grad_norm, snip, grasp, and Fisher get worse when scaled.

According to these results, naswot is the best choice when scaled since the Spearman rank correlation is perfect. It does not matter which random seed or mini-batch sampling is chosen, naswot scored the architectures in the same order every time.

Table 1. Mean Spearman rank correlation over multiple random seeds for each ZC proxy score z and its scaled variants by layers z_l and channels z_c.

	Grad_norm	**Snip**	**Grasp**	**Fisher**	**Synflow**	**Jacob_cov**	**Naswot**
z	0.739	0.809	0.491	0.321	0.793	0.473	**0.917**
z_l	0.395	0.454	0.308	0.209	0.824	0.983	**0.999**
z_c	0.605	0.535	0.377	0.217	0.841	0.987	**0.999**

2.6.2. Are ZC Proxies Robust with Regards to Architecture Size?

For the third research question, we compute ZC proxy scores of the 30 sampled architectures for different size configurations. We initialize each of the 30 architectures with the channel depth at first layer $c_1 \in \{4, 8\}$ and maximum channel depth throughout the architecture $c_{\max} \in \{32, 64, 128\}$, resulting in six different combinations. For each of the six size combination, we score the 30 architectures and rank them accordingly. Afterwards we compute the Spearman rank correlation between these rankings. Additionally, this experiment is repeated for 24 different random seeds. The results are shown in Table 2, where we show the mean Spearman rank correlation over the 6 hyperparameter combinations and 24 random seeds.

Again, scaled naswot shows the most robustness closely followed by scaled jacob_cov, hence, the choice of hyperparameters does not matter when using these two ZC proxies. All other ZC proxies are not robust with respect to hyperparameter choice, and therefore, if used, hyperparameters need to be chosen carefully when using these ZC proxies. We note that these results are also affected by the robustness of ZC proxies with respect to the initialization, i.e., low correlation between random seeds also results in low correlation with respect to hyperparameter choice.

Table 2. Mean Spearman rank correlation over multiple architecture sizes for each ZC proxy score z and its scaled variants by layers z_l and channels z_c.

	Grad_norm	Snip	Grasp	Fisher	Synflow	Jacob_cov	Naswot
z	0.741	0.809	0.492	0.325	0.787	0.475	**0.908**
z_l	0.371	0.361	0.296	0.213	0.822	0.983	**0.999**
z_c	0.773	0.707	0.520	0.258	0.857	0.990	**0.997**

2.6.3. Are ZC Proxies and Validation Loss Correlated?

To answer the fourth research question, we sample 161 random architectures from our search space. As for the third research question, we initialize each of the 161 architectures with the channel depth at first layer $c_1 \in \{4, 8\}$ and maximum channel depth throughout the architecture $c_{\max} \in \{32, 64, 128\}$, resulting in 966 total architectures. As mentioned before, to obtain the true fitness of each architecture, we train them until convergence three times for different batch sizes (32, 64, 128) and set the fitness to the best validation loss reached during training. We report the mean Spearman rank correlation of ZC proxy score and best validation loss (fitness) over all combinations in Table 3.

Overall, naswot performs the best out of all proxies. Snip and scaled jacob_cov perform approximately as well as naswot, while grasp, Fisher, and synflow are outperformed.

Table 3. Mean Spearman rank correlation of the best validation loss of each architecture and each ZC proxy score z and its scaled variants by layers z_l and channels z_c.

	Grad_norm	Snip	Grasp	Fisher	Synflow	Jacob_cov	Naswot
z	−0.655	−0.684	−0.484	−0.398	−0.252	−0.483	**−0.693**
z_l	0.015	−0.243	−0.064	−0.068	−0.052	−0.729	**0.737**
z_c	0.400	0.201	0.148	0.098	0.047	−0.732	**0.737**

To sum up, no ZC proxy is perfectly robust out of the box for our setting with regards to weight initialization, mini-batch sampling, mini-batch size, and architecture size. After scaling by the number of layers or channels in the architecture, naswot is robust and jacob_cov slightly worse. All other ZC proxies are not robust and therefore unusable for traffic prediction. Scaled naswot and scaled jacob_cov are the most correlated with validation loss. Combining the robustness and correlation results, naswot comes out as the best ZC proxy to use for our search space and task. The robustness with respect to architecture size makes it possible to run scaled naswot on very small versions of the architectures, further lowering computation cost.

2.7. Low Cost Evolutionary Neural Architecture Search

In this Section we incorporate the naswot ZC proxy into the genetic algorithm described in Section 2.4 and evaluate our method on four real world datasets. In the following we describe the four datasets, the evaluation metrics and baseline models we use for comparison.

2.7.1. Datasets

Our experiments are conducted on four real world datasets, two of which are concerned with traffic flow prediction and two with traffic speed prediction:

- PeMSD4–The PeMSD4 dataset is made up of traffic flow measurements from 307 loop detectors in the San Francisco Bay Area within the period from 1 January 2018 to 28 February 2018 [10].
- PeMSD8–The PeMSD8 dataset contains traffic flow measurements from 170 loop detectors in the San Bernardino Area from 1 July 2016 to 31 August 2016 [10].

- METR-LA–The METR-LA dataset includes traffic speed readings at 207 sensors located on the highways of Los Angeles County from 1 March 2012 to 30 June 2012 [31].
- PEMS-BAY–The PEMS-BAY dataset comprises traffic speed data from 325 measurement sites in the Bay Area of California from 1 January 2017 to 31 May 2017 [31].

All datasets are aggregated into 5 min windows, resulting in 288 timestamps per day. For training, the data are normalized by standard normalization for each node and feature. Given a timestamp t, we want to predict the next hour of traffic conditions, i.e., 12 timesteps. The input $\mathbf{X}_t \in \mathbb{R}^{N \times F \times 12}$ to our network is made up of a recent, daily, and weekly segment from the historical data. These segments are defined as follows:

$$\begin{aligned} \mathbf{X}_t^{\text{recent}} &= [X_{t-11}, ..., X_t] \\ \mathbf{X}_t^{\text{daily}} &= [X_{t+1-288}, ..., X_{t+12-288}] \\ \mathbf{X}_t^{\text{weekly}} &= [X_{t+1-7\times288}, ..., X_{t+12-7\times288}] \end{aligned}$$

As can be seen, the recent segment comprises the last hour of data, the daily segment includes data from the same hour to be predicted, but on the day before and the weekly segment contains the same hour we predict, but one week earlier. Additionally, we include data about time of the day and day of the week for the prediction segment. The segments and time information are stacked in the feature dimension of the input, i.e., $\mathbf{X}_t \in \mathbb{R}^{N \times 5 \times 12}$. Stacking in the feature dimension has not been done in previous works. In [8,32] multiple modules with the same architectures and a fusion layer are used, while in [10,15] only the recent segment is used. We have conducted experiments comparing different input methods and concluded that stacking multiple segments in the feature dimension works best for our approach. However, future research might be conducted to set a standard for the task of traffic prediction.

We seperate the datasets into training, validation and test sets with a 7-1-2 ratio. The adjacency matrices are constructed as in previous works by road network distance and Gaussian kernel threshholding [7].

We remark that, due to the choice of inputs, the resulting datasets include fewer samples than in other works, where only the last 12 timesteps are used as inputs [7,10,15]. Therefore, direct comparisons with their results are to be taken with caution. For fair comparison in this work, we evaluate the baseline models on our datasets.

2.7.2. Metrics

We use mean absolute error (MAE), rooted mean squared error (RMSE) and mean absolute percentage error (MAPE) to evaluate our framework and the baselines:

$$\begin{aligned} MAE &= \frac{1}{N}\sum_{i=1}^{N} \mid \hat{y}_i - y_i \mid, \\ RMSE &= \sqrt{\frac{1}{N}\sum_{i=1}^{N}(\hat{y}_i - y_i)^2}, \\ MAPE &= 100\% \times \frac{1}{N}\sum_{i=1}^{N} \frac{\mid \hat{y}_i - y_i \mid}{y_i} \end{aligned}$$

Here, $N, \hat{y}_i$ and y_i respectively refer to the number of samples, predicted values and ground truth values. Since y_i can be zero-valued for some measurements, we only compute MAPE when ground truth is larger than one.

2.7.3. Baselines

We compare our framework against the following models:

- Historical average (HA)–Traffic is modeled as a seasonal process. We predict future timesteps by taking the average over the last n_d (to be determined) days of the same time.
- AGCRN–Adaptive graph convolutional recurrent network captures spatio and temporal dependencies automatically from the data without the need of predefined adjacency matrices for the graph convolution [10].
- Graph WaveNet–Deploys WaveNet [33] and graph convolutions for modelling spatio-temporal graph signals. The adjacency matrix is self-adapting by discovering structures in the data without prior knowledge [7].
- AutoSTG–Gradient-based NAS framework for spatio-temporal prediction. Pan et al. [15] use special modules for capturing spatio-temporal dependencies from meta data of the attributed graph.

We evaluate all baselines on our own dataset as described in Section 2.7.1. To this end, we adapt the publicly available code and conduct the recommended hyperparameter search of each model.

2.7.4. Framework Settings

We apply GA on each of the four datasets until convergence of the algorithm. We have not performed extensive hyperparameter tuning, as we want to show that the algorithm can be applied by non-experts. We use a warmstart size of 1000 genomes to explore a large chunk of the search space in the beginning. This should lead to a high diversity in the population. Afterwards, we decrease the population size to 100 for a faster search time. Note that this is a big increase in population size to previous work [11]. The crossover probability is fixed to 0.9 while mutation probability p_m is adaptively set for each genome based on their rank in the interval [0.05, 0.15]:

$$p_m(A) = 0.15 - \frac{n_p - rank(A)}{n_p} \times 0.1$$

We use the naswot score to rank architectures. As shown in Section 2.6, the naswot score is stable with regards to batch size and architecture size. Hence, we can select them to be small, which will lead to faster search time. Therefore, each network is scored by naswot with a minibatch size of 32, maximum channel size of 32 and starting channel size of 4. We conduct each experiment on one Nvidia GeForce GTX 3090 GPU. The best model architecture is trained until convergence for different starting learning rates (0.02, 0.01, 0.005) and batch sizes (32, 64, 128). Finally, the best performing model on the validation set is used for measuring performance on the test set.

3. Results

The results of the prediction performance for the four datasets are presented in Table 4, where the MAE, RMSE, and MAPE for the 15 min, 30 min, and 60 min horizons are reported. The experiments were conducted twice with different random seeds, except for the LENAS experiments which were conducted thrice.

As anticipated, the simplest model, HA, showed the worst performance on all four datasets due to its inability to capture the complexity of the spatio-temporal data. This model can only capture the general trend of the data and fails to adapt to local changes in the trend.

On the other hand, the two hand-crafted deep learning models, AGCRN and GWN, which can model spatio-temporal dependencies, achieved better predictive performance. However, AGCRN had the worst performance among all deep learning models. GWN outperformed all other methods on the METR-LA dataset and achieved the best or comparable performance to AutoSTG and ENAS while slightly outperforming LENAS on the PEMS-BAY dataset.

We were unable to conduct experiments with AutoSTG on PeMSD4 and PeMSD8 due to the lack of metadata for these datasets. Nonetheless, AutoSTG was the best-performing method on the PEMS-BAY dataset for the 15 min horizon and exhibited comparable performance to other approaches but lacked stability with different random seeds.

The ENAS framework described in [23] outperformed all other approaches on PeMSD4 for all metrics and on most metrics on PeMSD8. However, its performance on METR-LA was underwhelming compared to GWN. For PEMS-BAY, the ENAS framework was able to keep up with GWN and AutoSTG.

As expected, LENAS was unable to outperform ENAS as it searches the architecture space less accurately. The results on METR-LA and PEMS-BAY were competitive with other deep learning models, while the performance was lacking on PeMSD8 and PeMSD4, especially for the shorter horizons.

Regarding search time, ENAS exhibited the worst performance, requiring approximately 300 GPU hours for the smallest dataset (PeMSD8) and approximately 1200 GPU hours for the largest dataset (PEMS-BAY). AutoSTG had to be run multiple times for different combinations of architecture-related hyperparameters, with each run taking around 10 GPU hours for search and 5 h for training the discovered architectures. AGCRN and GWN took the least time since they did not include an architecture search process. LENAS improved ENAS search time to 1–4 GPU hours depending on the dataset with a much larger population size.

Table 4. Traffic forecast performance on PeMSD4, PeMSD8, METR-LA and PEMS-BAY datasets. Here, GWN, ENAS and LENAS respectively denote Graph WaveNet, GA without ZC proxies and our framework.

	MAE			**RMSE**			**MAPE**		
PeMSD4	15 min	30 min	60 min	15 min	30 min	60 min	15 min	30 min	60 min
HA	34.33 ± 0.00	34.33 ± 0.00	34.33 ± 0.00	53.27 ± 0.00	53.27 ± 0.00	53.27 ± 0.00	$24.22\% \pm 0.00\%$	$24.22\% \pm 0.00\%$	$24.22\% \pm 0.00\%$
AGCRN	19.02 ± 0.07	19.88 ± 0.11	21.05 ± 0.36	30.99 ± 0.49	32.66 ± 0.53	34.56 ± 0.12	$12.49\% \pm 0.33\%$	$12.92\% \pm 0.32\%$	$13.92\% \pm 0.25\%$
GWN	18.28 ± 0.04	19.24 ± 0.06	20.95 ± 0.04	29.44 ± 0.09	31.09 ± 0.16	33.83 ± 0.21	$11.98\% \pm 0.03\%$	$12.60\% \pm 0.02\%$	$13.71\% \pm 0.00\%$
ENAS	$\mathbf{17.95 \pm 0.01}$	$\mathbf{18.77 \pm 0.00}$	$\mathbf{20.44 \pm 0.02}$	$\mathbf{29.22 \pm 0.01}$	$\mathbf{30.67 \pm 0.00}$	$\mathbf{33.21 \pm 0.03}$	$\mathbf{11.55\% \pm 0.01\%}$	$\mathbf{12.04\% \pm 0.03\%}$	$\mathbf{13.22\% \pm 0.03\%}$
LENAS	18.42 ± 0.04	19.34 ± 0.06	20.81 ± 0.10	29.62 ± 0.05	31.14 ± 0.07	33.42 ± 0.11	$11.82\% \pm 0.04\%$	$12.43\% \pm 0.05\%$	$13.43\% \pm 0.08\%$
PeMSD8	15 min	30 min	60 min	15 min	30 min	60 min	15 min	30 min	60 min
HA	31.33 ± 0.00	31.33 ± 0.00	31.33 ± 0.00	48.72 ± 0.00	48.72 ± 0.00	48.72 ± 0.00	$23.50\% \pm 0.00\%$	$23.50\% \pm 0.00\%$	$23.50\% \pm 0.00\%$
AGCRN	13.17 ± 0.08	13.67 ± 0.13	14.88 ± 0.23	22.34 ± 0.12	23.66 ± 0.15	25.67 ± 0.19	$8.46\% \pm 0.08\%$	$8.81\% \pm 0.11\%$	$9.73\% \pm 0.24\%$
GWN	13.69 ± 0.15	14.18 ± 0.08	14.98 ± 0.08	21.90 ± 0.13	$\mathbf{23.18 \pm 0.05}$	$\mathbf{25.03 \pm 0.05}$	$8.81\% \pm 0.15\%$	$9.17\% \pm 0.09\%$	$9.94\% \pm 0.02\%$
ENAS	$\mathbf{13.14 \pm 0.01}$	$\mathbf{13.62 \pm 0.17}$	$\mathbf{14.85 \pm 0.18}$	$\mathbf{21.77 \pm 0.07}$	23.18 ± 0.25	25.27 ± 0.15	$\mathbf{8.33\% \pm 0.07\%}$	$\mathbf{8.68\% \pm 0.14\%}$	$\mathbf{9.64\% \pm 0.07\%}$
LENAS	14.19 ± 0.01	14.89 ± 0.07	15.85 ± 0.04	22.48 ± 0.01	23.97 ± 0.10	25.65 ± 0.02	$8.88\% \pm 0.03\%$	$9.35\% \pm 0.05\%$	$10.21\% \pm 0.01\%$
METR-LA	15 min	30 min	60 min	15 min	30 min	60 min	15 min	30 min	60 min
HA	13.66 ± 0.00	13.66 ± 0.00	13.66 ± 0.00	21.28 ± 0.00	21.28 ± 0.00	21.28 ± 0.00	$19.82\% \pm 0.00\%$	$19.82\% \pm 0.00\%$	$19.82\% \pm 0.00\%$
AGCRN	3.38 ± 0.00	4.07 ± 0.00	5.04 ± 0.00	7.48 ± 0.00	9.28 ± 0.00	11.34 ± 0.00	$8.46\% \pm 0.00\%$	$10.39\% \pm 0.00\%$	$12.90\% \pm 0.00\%$
GWN	$\mathbf{2.84 \pm 0.01}$	$\mathbf{3.22 \pm 0.01}$	$\mathbf{3.62 \pm 0.04}$	$\mathbf{5.45 \pm 0.03}$	$\mathbf{6.44 \pm 0.00}$	$\mathbf{7.39 \pm 0.05}$	$\mathbf{7.40\% \pm 0.05\%}$	$\mathbf{8.67\% \pm 0.14\%}$	$\mathbf{10.14\% \pm 0.20\%}$
AutoSTG	3.05 ± 0.24	3.69 ± 0.41	4.60 ± 0.77	5.73 ± 0.29	7.21 ± 0.53	9.02 ± 1.06	$7.67\% \pm 0.48\%$	$9.56\% \pm 0.86\%$	$11.93\% \pm 1.43\%$
ENAS	2.97 ± 0.00	3.40 ± 0.01	3.88 ± 0.01	5.75 ± 0.02	6.78 ± 0.01	7.80 ± 0.01	$7.90\% \pm 0.04\%$	$9.50\% \pm 0.07\%$	$11.27\% \pm 0.07\%$
LENAS	3.06 ± 0.06	3.54 ± 0.06	4.00 ± 0.04	5.94 ± 0.12	7.07 ± 0.20	8.15 ± 0.17	$8.14\% \pm 0.27\%$	$9.85\% \pm 0.23\%$	$11.45\% \pm 0.13\%$
PEMS-BAY	15 min	30 min	60 min	15 min	30 min	60 min	15 min	30 min	60 min
HA	3.28 ± 0.00	3.28 ± 0.00	3.28 ± 0.00	6.54 ± 0.00	6.54 ± 0.00	6.54 ± 0.00	$7.99\% \pm 0.00\%$	$7.99\% \pm 0.00\%$	$7.99\% \pm 0.00\%$
AGCRN	1.41 ± 0.03	1.72 ± 0.01	1.99 ± 0.01	2.95 ± 0.02	3.89 ± 0.01	4.56 ± 0.02	$3.09\% \pm 0.01\%$	$3.99\% \pm 0.00\%$	$4.79\% \pm 0.00\%$
GWN	1.33 ± 0.01	$\mathbf{1.62 \pm 0.02}$	$\mathbf{1.90 \pm 0.02}$	2.81 ± 0.02	3.71 ± 0.04	$\mathbf{4.44 \pm 0.11}$	$2.84\% \pm 0.01\%$	$3.74\% \pm 0.04\%$	$4.59\% \pm 0.12\%$
AutoSTG	1.32 ± 0.05	1.63 ± 0.07	1.93 ± 0.09	2.80 ± 0.10	3.78 ± 0.15	4.59 ± 0.21	$2.82\% \pm 0.15\%$	$3.80\% \pm 0.26\%$	$4.71\% \pm 0.32\%$
ENAS	1.32 ± 0.00	1.63 ± 0.00	1.91 ± 0.00	2.81 ± 0.00	$\mathbf{3.71 \pm 0.01}$	4.45 ± 0.01	$2.82\% \pm 0.00\%$	$3.74\% \pm 0.01\%$	$4.59\% \pm 0.00\%$
LENAS	1.36 ± 0.00	1.68 ± 0.00	1.95 ± 0.01	2.87 ± 0.01	3.81 ± 0.02	4.53 ± 0.03	$2.93\% \pm 0.01\%$	$3.89\% \pm 0.01\%$	$4.67\% \pm 0.01\%$

4. Discussion and Conclusions

In this research, we explored zero-cost proxies, namely the naswot proxy, to estimate network performance for traffic forecasting tasks. Our novel approach centered on utilizing a performance estimation process, rather than training until convergence, as typically employed in other frameworks such as ENAS.

We observed that our LENAS framework, despite its advantage of fast search times and lower computational costs, displayed worse performance compared to other deep learning models. This underperformance is largely attributed to the disconnect between the naswot score and the validation loss of the architectures. Our results indicated an average Spearman rank correlation of 0.737, as highlighted in Table 3, signifying a substantial margin of error when ranking architectures using the naswot score as opposed to the validation loss.

The lack of correlation between the naswot score and the validation loss was a primary factor contributing to the inaccuracies in our model. Hence, while the naswot proxy, once normalized by network size, proved to be stable concerning weight initialization, mini-batch sampling, and size, its use revealed notable challenges in achieving comparable accuracy with the baseline models that do not incorporate performance estimation.

The experimental trials conducted with two traffic speed benchmarks and two traffic flow benchmarks affirmed the double-edged nature of using the naswot score. On one hand, we managed to speed up the neural architecture search process by two orders of magnitude and explore the architecture space more thoroughly. Conversely, this came at the cost of a decrease in accuracy, which emphasizes the need to balance speed with precision in the application of such zero-cost proxies. When analyzing the experimental results in Table 4, LENAS often performs worse than GWN, a handcrafted approach, suggesting that the inclusion of naswot as performance estimator might be less effective than using GWN. Both methods use adjacency matrices for the graph convolution. The choice of the adjacency matrix can be crucial for the performance of the neural network. LENAS employs a predefined adjacency matrix, while GWN uses an adaptive matrix which adapts to the data at hand. This might be beneficial for the GWN approach and, hence, in future research, LENAS should be extended to include adaptive adjacency matrices or attention mechanisms [34].

In light of our findings, future research endeavors should prioritize designing zero-cost proxies specifically geared towards regression tasks to yield more accurate results. While our LENAS framework showed potential in terms of reduced search time and low computational requirements, the accuracy of the naswot proxy needs further enhancement.

Overall, the exploration of zero-cost proxies such as the naswot score has shown the potential and challenges of such an approach. This work opens up new pathways for evolutionary neural architecture search processes, especially in the field of traffic forecasting, provided the inaccuracies are effectively addressed in future iterations.

Author Contributions: Conceptualization, D.K. and C.B.; methodology, D.K.; software, D.K.; validation, D.K.; formal analysis, D.K.; investigation, D.K.; resources, D.K.; data curation, D.K.; writing—original draft preparation, D.K.; writing—review and editing, D.K.; visualization, D.K.; supervision, C.B.; project administration, D.K.; funding acquisition, C.B. All authors have read and agreed to the published version of the manuscript.

Funding: This work was funded by the European Regional Development Fund (ERDF).

Data Availability Statement: We only used publicly available data, see Section 2.7.1.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Sun, H.; Liu, H.X.; Xiao, H.; He, R.R.; Ran, B. Use of Local Linear Regression Model for Short-Term Traffic Forecasting. *Transp. Res. Rec.* **2003**, *1836*, 143–150. [CrossRef]
2. Makridakis, S.; Hibon, M. ARMA Models and the Box–Jenkins Methodology. *J. Forecast.* **1997**, *16*, 147–163. [CrossRef]

3. Zivot, E.; Wang, J. Vector Autoregressive Models for Multivariate Time Series. In *Modeling Financial Time Series with S-Plus*; Springer: New York, NY, USA, 2003.
4. Mallek, A.; Klosa, D.; Büskens, C. Impact of Data Loss on Multi-Step Forecast of Traffic Flow in Urban Roads Using K-Nearest Neighbors. *Sustainability* **2022**, *14*, 11232. [CrossRef]
5. Mallek, A.; Klosa, D.; Büskens, C. Enhanced K-Nearest Neighbor Model For Multi-steps Traffic Flow Forecast in Urban Roads. In Proceedings of the 2022 IEEE International Smart Cities Conference (ISC2), Pafos, Cyprus, 26–29 September 2022; pp. 1–5. [CrossRef]
6. Fu, R.; Zhang, Z.; Li, L. Using LSTM and GRU neural network methods for traffic flow prediction. In Proceedings of the 2016 31st Youth Academic Annual Conference of Chinese Association of Automation (YAC), Wuhan, China, 11–13 November 2016; pp. 324–328. [CrossRef]
7. Wu, Z.; Pan, S.; Long, G.; Jiang, J.; Zhang, C. Graph WaveNet for Deep Spatial-Temporal Graph Modeling. In Proceedings of the IJCAI, Macao, 10–16 August 2019.
8. Ge, L.; Li, S.; Wang, Y.; Chang, F.; Wu, K. Global Spatial-Temporal Graph Convolutional Network for Urban Traffic Speed Prediction. *Appl. Sci.* **2020**, *10*, 1509. [CrossRef]
9. Klosa, D.; Mallek, A.; Büskens, C. Short-Term Traffic Flow Forecast Using Regression Analysis and Graph Convolutional Neural Networks. In Proceedings of the 2021 IEEE 23rd Int Conf on High Performance Computing & Communications; 7th Int Conf on Data Science & Systems; 19th Int Conf on Smart City; 7th Int Conf on Dependability in Sensor, Cloud & Big Data Systems & Application (HPCC/DSS/SmartCity/DependSys), Haikou, China, 20–22 December 2021; pp. 1413–1418. [CrossRef]
10. Bai, L.; Yao, L.; Li, C.; Wang, X.; Wang, C. Adaptive Graph Convolutional Recurrent Network for Traffic Forecasting. In Proceedings of the NIPS'20': 34th International Conference on Neural Information Processing Systems, Vancouver, BC, Canada, 6–12 December 2020.
11. Pham, H.; Guan, M.; Zoph, B.; Le, Q.; Dean, J. Efficient Neural Architecture Search via Parameters Sharing. In Proceedings of the 35th International Conference on Machine Learning, Stockholm, Sweden, 10–15 July 2018; Machine Learning Research; Dy, J., Krause, A., Eds.; Volume 80, pp. 4095–4104.
12. Liu, H.; Simonyan, K.; Yang, Y. DARTS: Differentiable Architecture Search. In Proceedings of the International Conference on Learning Representations, New Orleans, LA, USA, 6–9 May 2019.
13. Gao, Y.; Yang, H.; Zhang, P.; Zhou, C.; Hu, Y. Graph Neural Architecture Search. In Proceedings of the IJCAI'20: Twenty-Ninth International Joint Conference on Artificial Intelligence, Online, 7–15 January 2021.
14. Zhou, K.; Song, Q.; Huang, X.; Hu, X. Auto-GNN: Neural Architecture Search of Graph Neural Networks. *arXiv* **2019**, arXiv:1909.03184.
15. Pan, Z.; Ke, S.; Yang, X.; Liang, Y.; Yu, Y.; Zhang, J.; Zheng, Y. AutoSTG: Neural Architecture Search for Predictions of Spatio-Temporal Graph. In Proceedings of the WWW '21: Web Conference 2021, New York, NY, USA, 19–23 April 2021; pp. 1846–1855. [CrossRef]
16. Zoph, B.; Le, Q. Neural Architecture Search with Reinforcement Learning. In Proceedings of the International Conference on Learning Representations, Toulon, France, 24–26 April 2017.
17. Elsken, T.; Metzen, J.H.; Hutter, F. Efficient Multi-Objective Neural Architecture Search via Lamarckian Evolution. In Proceedings of the 7th International Conference on Learning Representations, New Orleans, LA, USA, 6–9 May 2019.
18. Lopes, V.; Alirezazadeh, S.; Alexandre, L.A. EPE-NAS: Efficient Performance Estimation Without Training for Neural Architecture Search. In *Artificial Neural Networks and Machine Learning–ICANN 2021*; Springer: Berlin/Heidelberg, Germany, 2021; pp. 552–563. [CrossRef]
19. White, C.; Zela, A.; Ru, B.; Liu, Y.; Hutter, F. How Powerful are Performance Predictors in Neural Architecture Search? *Adv. Neural Inf. Process. Syst.* **2021**, *34*, 28454–28469. [CrossRef]
20. Vlahogianni, E.I.; Karlaftis, M.G.; Golias, J.C. Optimized and meta-optimized neural networks for short-term traffic flow prediction: A genetic approach. *Transp. Res. Part C Emerg. Technol.* **2005**, *13*, 211–234. .: 10.1016/j.trc.2005.04.007. [CrossRef]
21. Rahimipour, S.; Moienfar, R.; Hashemi, S.M. Traffic Prediction Using a Self-Adjusted Evolutionary Neural Network. *J. Mod. Transp.* **2018**, *27*, 306–316. [CrossRef]
22. Li, L.; Qin, L.; Qu, X.; Zhang, J.; Wang, Y.; Ran, B. Day-ahead traffic flow forecasting based on a deep belief network optimized by the multi-objective particle swarm algorithm. *Knowl.-Based Syst.* **2019**, *172*, 1–14. [CrossRef]
23. Klosa, D.; Büskens, C. Evolutionary Neural Architecture Search for Traffic Forecasting. In Proceedings of the 21st IEEE International Conference on Machine Learning and Applications, to Appear in IEEE Xplore, Nassau, Bahamas, 12–15 December 2022; pp. 1230–1237.
24. Yu, F.; Koltun, V. Multi-Scale Context Aggregation by Dilated Convolutions. In Proceedings of the International Conference on Learning Representations (ICLR), San Juan, Puerto Rico, USA, 2–4 May 2016.
25. Abdelfattah, M.S.; Mehrotra, A.; Dudziak, L.; Lane, N.D. Zero-Cost Proxies for Lightweight NAS. *arXiv* **2021**. [CrossRef]
26. Lee, N.; Ajanthan, T.; Torr, P.H.S. SNIP: Single-shot Network Pruning based on Connection Sensitivity *arXiv* **2018**. [CrossRef]
27. Wang, C.; Zhang, G.; Grosse, R. Picking Winning Tickets Before Training by Preserving Gradient Flow. *arXiv* **2020**. [CrossRef]
28. Tanaka, H.; Kunin, D.; Yamins, D.L.K.; Ganguli, S. Pruning neural networks without any data by iteratively conserving synaptic flow. *arXiv* **2020**. [CrossRef]

29. Theis, L.; Korshunova, I.; Tejani, A.; Huszár, F. Faster gaze prediction with dense networks and Fisher pruning. *arXiv* **2018**. [CrossRef]
30. Mellor, J.; Turner, J.; Storkey, A.; Crowley, E.J. Neural Architecture Search without Training. *arXiv* **2020**. [CrossRef]
31. Li, Y.; Yu, R.; Shahabi, C.; Liu, Y. Diffusion Convolutional Recurrent Neural Network: Data-Driven Traffic Forecasting. In Proceedings of the International Conference on Learning Representations, Vancouver, BC, Canada, 30 April–3 May 2018.
32. Guo, S.; Lin, Y.; Feng, N.; Song, C.; Wan, H. Attention Based Spatial-Temporal Graph Convolutional Networks for Traffic Flow Forecasting. In Proceedings of the AAAI Conference on Artificial Intelligence, Honolulu, HI, USA, 27 January–1 February 2019; Volume 33, pp. 922–929. [CrossRef]
33. van den Oord, A.; Dieleman, S.; Zen, H.; Simonyan, K.; Vinyals, O.; Graves, A.; Kalchbrenner, N.; Senior, A.; Kavukcuoglu, K. WaveNet: A Generative Model for Raw Audio. In Proceedings of the 9th ISCA Workshop on Speech Synthesis Workshop (SSW 9), Sunnyvale, CA, USA, 13–15 September 2016; p. 125.
34. Cai, L.; Janowicz, K.; Mai, G.; Yan, B.; Zhu, R. Traffic transformer: Capturing the continuity and periodicity of time series for traffic forecasting. *Trans. GIS* **2020**, *24*, 736–755. [CrossRef]